Raspberry Pi Security System

Introduction

For months I've been wanting to build something with the Raspberry Pi - a $35 Linux computer. After considering sprinkler systems that take into account weather forecasts, media center to download and manage videos, and home automation - I finally settled on replacing my $200 a year alarm service with a Raspberry Pi server and an android app.

The basic idea is when our home alarm is tripped, a message is sent to my android phone and I can listen to the alarm and notify either the neighbors or the authorities.

I also added a webcam with streaming to watch the front.

In the future, I plan to send a message to Android when the door bell button is pressed.

The overall project took me about 3 weeks of 12 hours per week to complete - most of which was consumed by learning Python and writing the software - all of which is now complete and open sourced for anyone wishing to do the same!

Project Plan

The overall project required the following activities:

- Wire the existing alarm system to the Raspberry Pi so that tripping the alarm will signal a Python script.
- Write a Python script for the Raspberry Pi to perform the following:
    - Watch for any alarm signals and send a message to the Android phone.
    - Forward information on the audio and video streams so that the Android phone can watch and listen in addition to the alarm.
    - Log any events.
- Write an Android app that receives alerts and plays the audio and video streams from the Raspberry Pi.
- Write a relay server for the Android alerts - Since the Android app uses Google Cloud Messaging (GCM) and GCM only accepts messages from fixed IP address, the Raspberry Pi will send message to a known fixed IP address which will the forward the message onto GCM. This allows the Pi to be installed in a home network with the external IP address is not guaranteed.
- Install software on the Pi which will stream audio and video from a webcam (which happens to be lying around unused).
- Write cron jobs and scripts on the Pi to do the following:
    - Launch the Python notification scripts on boot
    - Launch the video streaming on boot

    - Launch the audio streaming on boot
    - Archive video and image from the webcam in folders named by the dates
    - Delete old archived files ones the number of days or storage space is exceeded [TODO]
- Configure a server on Amazon Web Service to run the relay server
- Install the webcam, Pi, networking, power, and wires to the alarm system
- Install an FTP server to download the images from the webcam recording

Initial Setup of the Pi

I ordered a bundle from Amazon which included a Raspberry Pi B+, 8GB micro SD with pre-installed OS, case, wi-fi chip, HDMI cable and heat-sinks for $60. In hindsight, I could have saved some $$ just getting the B+, building my own case and using a larger Micros SD card.

Unfortunately, I broke the provided SD card by trying to install the case after I installed the SD card.

I bought a 32GB micro SD card from Fry's on sale for $17. The extra memory came in handy for storing movies and images from the webcam. I could have easily gotten away with a 16GB card, but it was only $4 more for the 32GB card. I made sure it was a high speed class 10 card.

The image was downloaded from the official Raspberry Pi download site.

I chose the Raspbian Debian Wheezy (release date 2014-09-09) image.

To load the image on the SD card, I downloaded the Win32DiskImager utility from the Sourceforge Project page to my Windows 8 laptop. Basically following the instructions provided by Raspberry Pi.

To do the initial install, I connected an HDMI cable from the Pi to my TV and connected a USB mouse and keyboard to the Pi.

On boot, I initialized the timezone, date, etc.

I also changed the default password for the pi user.

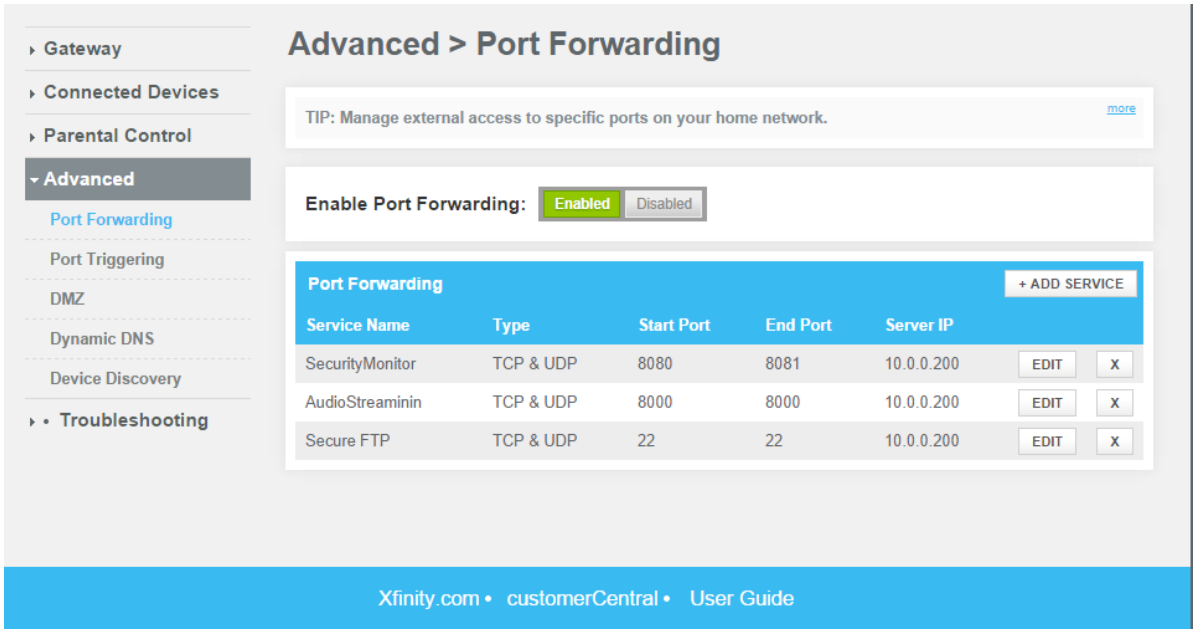
Network Configuration

For the network, I assigned a fixed IP address. This made it easier to ssh into the device from my laptop.

Below are the steps I used to configure the network:

- Set static IP address on the raspberry PI to x.x.x.200 by changing the following in the /etc/network/interfaces file:
    - iface eth0 inet static
    - address 10.0.0.200
    - netmask 255.255.255.0
    - network 10.0.0.0
    - gateway 10.0.0.1

- Add the static IP to the router static IP configuration
- To allow access to the webcam, secure ftp and audio streams from outside the home network, I added port forwarding for ports 8080, 8081, and 22 on IP address 10.0.0.200 in the router:



Setting up ssh

After rebooting and ensuring the network works, I tested access using ssh. I used the putty client on windows.

Access is pretty straightforward. For the host, use pi@10.0.0.200. I left the default port to 22.

Once I verified I could use ssh, I disconnected the Pi from the TV.

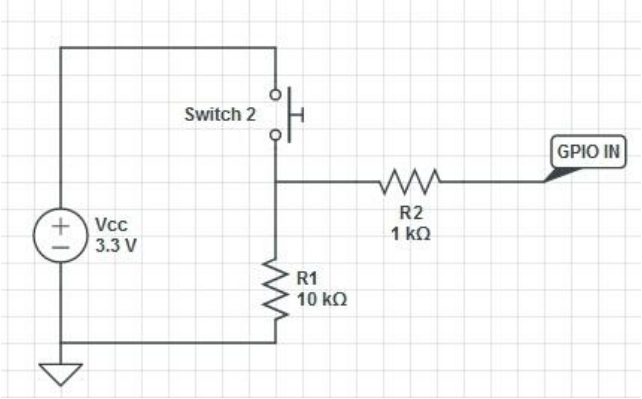
Wiring to Existing Alarm System

The alarm system is a NAMCO XXXX. The wiring diagram shows a relay which should trigger when the alarm is tripped.

Below are the steps I used to wire to the Pi model B+

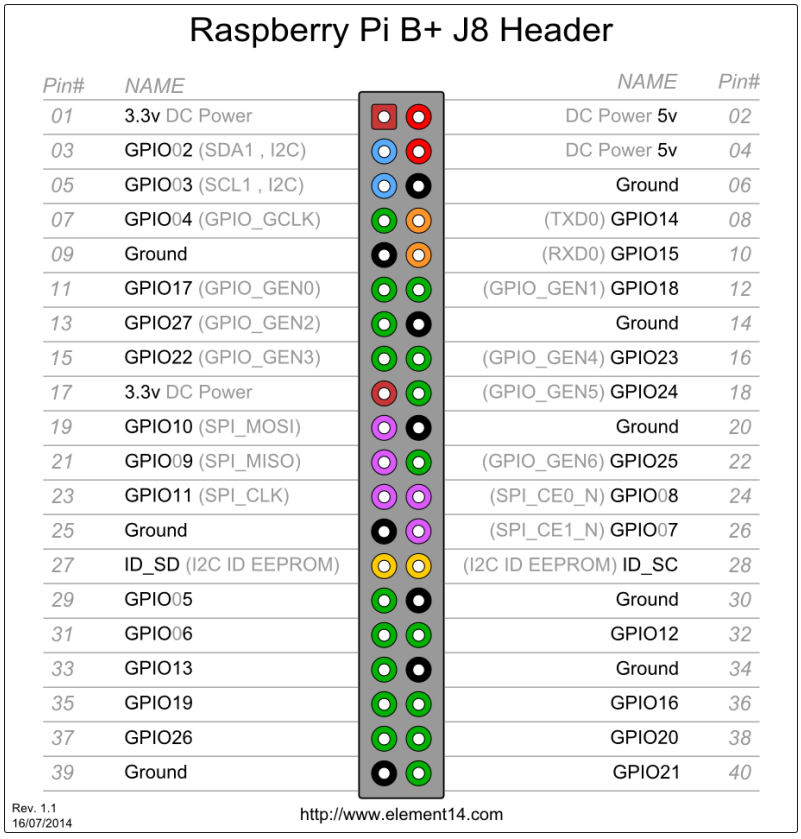
Note: The wiring pins are different between different versions of the Pi.

1. Run wires from the relay terminal C and relay terminal N/O in the alarm box.
2. When the alarm goes off, these should close like a switch. **Verify that no voltage goes across the wire**. The instruction talk about a configuration where 24V may actually go across the relay.
3. Wire the relay as in switch 2 in the diagram below from the Physical Computing with Raspberry Pi article:

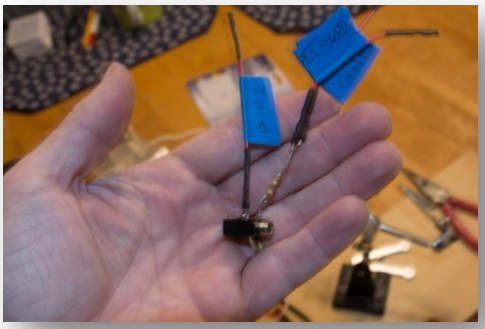


4. Pin 1 is the + 3.3V
5. Pin 39 is ground
6. Pin 11 is GPIO 17 (wiringPi pin 0, BCM Pin 17)

The following is a pinout diagram from pi.gadgetoid.com:



I soldered the resistors to a 3.5mm headphone female jack and attached the jack to the top of the case.



Physical Installation

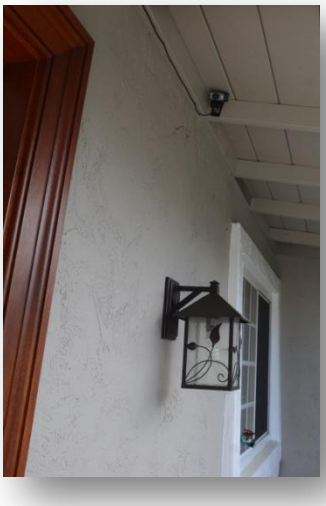
I chose the furnace closet to install the Pi. It had power and wasn't too far from the network router. It was also a few feet away from the front door for the web cam.

I ran bell wire from the Pi to the alarm (which was in a different room).

Instead of putting a power strip in the furnace room, I replaced the 2 receptacle 120v power with a 4 receptacle power - a 1/2 day of electrical work.

It was a short run for the network cable through the attic down to the room where the it was plugged into a switch port connected to the router.

The webcam was installed outside with a USB extension cable going through an attic vent and down to the Pi.



Alarm Software

All of the software I wrote for this project are available on Github under the Apache 2.0 software license.

There are 2 applications:

- SAHomeMonitor - The Android application to receive the alerts.

- HomeMonitorPiServer - Server side application. This actually consists of 2 application, the Python script that runs on the Pi to monitor the alarm and an optional Python relay script to run on a machine with a fixed IP address.

The applications use Google Cloud Message for the notifications.

SAHomeMonitor

The client side application includes functionality to listen for alarm notifications, display the webcam live stream, and send the audio from the webcam to the phone speakers.

Currently, the binaries are not provided on the Google Store. If there is interest in this, please email me.

To compile and run the software, you will need an Android Development Environment (I used the Eclipse plugins) and support for both Java and C. See the readme file in the source code for details on other setup pre-requisites.

The application includes source code from Google GCM for messaging support, and Simple JPegView for displaying the webcam output.

HomeMonitorPiServer

The server side applications are written in Python, the preferred language of the Pi.

There are 2 modes to run the server, standalone and through a relay.

The file samonitor.py contains a variable use_gcm_relay. If set to true, the relay will be used. Additional configuration information is documented in the source code.

To run the script on the Python, download and run the samonitorserver with a single parameter containing the API key obtained from Google.

To run on boot, I added a script to run the homemonitor app in rc.local.

HomeMonitor Script

[INSERT SCRIPT TEXT]

HomeMonitorPiServer contains software from python-gcm for the Google Cloud Messaging API and Python Onetime Password library for authentication support.

Examples from the Physical Computing with Raspberry Pi article were referenced when writing the code to read the alarm status.

Setting up Web Camera Software

The webcam provides 3 functions:

- Live streaming of the webcam video to the Android app

- Recording still images when any motion occurs
- Recording videos when any motion occurs

To setup the webcam software, I pretty much followed the step by step directions from: https://medium.com/@Cvrsor/how-to-make-a-diy-home-alarm-system-with-a-raspberry-pi-and-a-webcam-2d5a2d61da3d

Below is a paste of the instructions from that site which were not already done in the above Pi setup:

Step #3: Setup motion

First you need to use **rpi-update** to add to your raspbian image the initially-missing UVC support:

```
sudo apt-get install rpi-update
sudo rpi-update
```

Next you need to upgrade your packages:

```
sudo apt-get update
sudo apt-get upgrade
```

Then you can install motion:

```
sudo apt-get install motion
```

Now if you run

```
lsusb
```

you should see your camera listed as a usb device, like so:

```
Bus 001 Device 006: ID 046d:0825 Logitech, Inc. Webcam C270
```

(If not then perhaps your webcam is not compatible with pi)

Next we proceed to configure motion:

```
sudo nano /etc/motion/motion.conf
```

This is a configuration file where you get to define parameters such as the port which motion will run on, or actions that will be triggered when movement is detected.

Here's a list of the parameters you most likely would want to configure:

- *daemon*: set to ON to start motion as a daemon service when pi boots,
- *webcam_localhost*: set to OFF so that you can access motion from other computers,
- *stream_port*: the port for the video stream (default 8081),
- *control_localhost*: set to OFF to be able to update parameters remotely via the web config interface,
- *control_port*: the port that you will access the web config interface (default 8080),
- framerate: number of frames per second to be captured by the webcam. Warning: setting above 5 fps will hammer your pi's performance!
- *post_capture*: specify the number of frames to be captured after motion has been detected.

You also need to edit the following file if you want to run motion as a daemon service:

```
sudo nano /etc/default/motion
```

and set start_motion_daemon to YES:

```
start_motion_daemon=yes
```

Then start motion by typing:

```
sudo service motion start
```

Wait for about 30 seconds for motion to start and then open the video stream from VLC or a similar program that can show video streams. If you use VLC player go to File>Open Network and enter the IP address of your pi followed by the stream_port, for example:

192.168.1.5:8083

In addition to the steps on the website, I made the additional changes to etc/default/motion specific to my implementation:

- Change resolution from default to 720X480. I had to change in the config file (changing on the web did not work).
- Added a script (archive-capture) file for on_move_end and on_image_save to move the files to an archive file.

Archive Capture Script

```
#!/bin/bash
# Script to move a file to the capture video archive
# intended to be run in the MOTION image capture utility
# Usage: archive-capture file
```

```
if [ $# -eq 0 ]; then
      echo "missing filename parmater"
      exit 1
fi
CAPTURE_ARCHIVE_DIR=/var/capture-archive
FILE_DATE=$(date -d "$(stat -c %y $1)" +%Y-%m-%d)
DEST_DIR=$CAPTURE_ARCHIVE_DIR
DEST_DIR+="/"
DEST_DIR+="$FILE_DATE"
if ! [ -d $DEST_DIR ]; then
      mkdir $DEST_DIR
      chmod 777 $DEST_DIR
fi
mv $1 $DEST_DIR
```

Clean Images Cron Job

To make sure we don't run out of memory, I wrote and installed a cron job (a Linux script which runs at regular intervals) named cleanimages to delete old archived videos and images.

To add the script as a cronjob, enter crontab -e and enter the following:

0 0 3 * * /home/pi/scripts/cleanimages > /dev/null

assuming the script below is in a file /home/pi/scripts/cleanimages. Don't forget to set the execute permission on the script (chmod +x /home/pi/scripts/cleanimages).

Clean Images Script

```
#!/bin/bash
# Cleans the a directory of files organized by subdirectories
# named by date in the form yyyy-mm-dd
SIZELIMIT=300000 # Number of bytes after which files will be
deleted
KEEPDAYS=60           # Number of days to keep files
DIRECTORY=/var/capture-archive
DIRSIZE=$(du -s $DIRECTORY | awk '{print $1}')
NUMDAYS=$(ls $DIRECTORY | wc -w)
# echo dirsize=$DIRSIZE
# echo numdays=$NUMDAYS
if ([ "$DIRSIZE" -gt "$SIZELIMIT" ]) || ([ "$NUMDAYS" -gt
"$KEEPDAYS" ])
  then
    for f in `ls $DIRECTORY`; do
    SUBDIR=$DIRECTORY/$f
#    echo subdir=$SUBDIR
    SUBDIRSIZE=$(du -s $SUBDIR | awk '{print $1}')
#    echo subdirsize=$SUBDIRSIZE
    NUMDAYS=$((NUMDAYS-1))
```

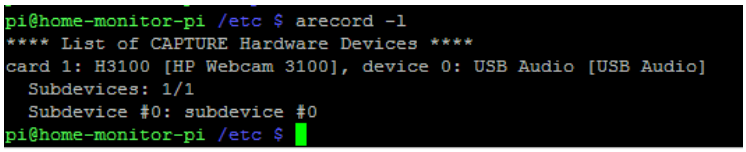
```
#     echo numdays=$NUMDAYS
    DIRSIZE=$(($DIRSIZE - $SUBDIRSIZE))
#     echo Dirsize = $DIRSIZE
    rm -rf $SUBDIR
    if ([ "$DIRSIZE" -le "$SIZELIMIT" ]) && ([ "$NUMDAYS" -le
"$KEEPDAYS" ])
      then
        break
    fi
done
fi
```

Add Audio

I used darkice and icecast2 to stream the audio from the Pi.

The setup for darkice and icecast2 was based on the live streaming mp3 audio with darkice and icecast2 on raspberry pi article.

I used the audio captured from the webcam audio device:



Installing Icecast2 (from blog post)

If you don't have an icecast2 server running somewhere, you could set one up on RasPi.

```
$ sudo aptitude install icecast2
```

The package manager will ask you to configure Icecast2. You should do so and set a hostname and passwords for source, relay and administration. Needless to say to use strong passwords. The source password will be needed in the darkice.cfg configuration. (See SOURCE_PASSWORD in the above example)

Configuring Icecast2

Edit the /etc/icecast2/icecast.xml.

I updated the following to reduce latency:

- <burst-on-connect>0</burst-on-connect>

Compiling and Installing Darkice (from the blog post)

The default darkice package comes without mp3 support. Since mp3 is the most widespread codec, I decided to build my own package of darkice with mp3 support.

Add a deb-src repository to your sources list at /etc/apt/sources.list:

```
$ sudo sh -c "echo 'deb-src http://mirrordirector.raspbian.org/raspbian/
wheezy main contrib non-free rpi' >> /etc/apt/sources.list"
$ sudo apt-get update
(...)
```

To fullfill the build dependencies, you have to install some additional packages:

```
$ sudo apt-get --no-install-recommends install build-essential
devscripts autotools-dev fakeroot dpkg-dev debhelper autotools-dev dh-
make quilt ccache libsamplerate0-dev libpulse-dev libaudio-dev lame
libjack-jackd2-dev libasound2-dev libtwolame-dev libfaad-dev libflac-dev
libmp4v2-dev libshout3-dev libmp3lame-dev
(...)
```

(Not sure if you need all these packages. But it won't hurt.)

Create a working directory:

```
$ mkdir src && cd src/
```

Get the source package of darkice:

```
$ apt-get source darkice
(...)
```

Change the compile configuration to match Raspbian environment:

```
$ cd darkice-1.0/
$ vi debian/rules
```

```
#!/usr/bin/make -f

%:
	dh $@

.PHONY: override_dh_auto_configure
override_dh_auto_configure:
	ln -s /usr/share/misc/config.guess .
	ln -s /usr/share/misc/config.sub .
	dh_auto_configure -- --prefix=/usr --
sysconfdir=/usr/share/doc/darkice/examples --with-vorbis-
prefix=/usr/lib/arm-linux-gnueabihf/ --with-jack-prefix=/usr/lib/arm-
linux-gnueabihf/ --with-alsa-prefix=/usr/lib/arm-linux-gnueabihf/ --
with-faac-prefix=/usr/lib/arm-linux-gnueabihf/ --with-aacplus-
prefix=/usr/lib/arm-linux-gnueabihf/ --with-samplerate-
prefix=/usr/lib/arm-linux-gnueabihf/ --with-lame-prefix=/usr/lib/arm-
linux-gnueabihf/ CFLAGS='-march=armv6 -mfpu=vfp -mfloat-abi=hard'
```

Please consider to download the rules file from here instead of copy&paste the lines above. The build will fail if line beginnings contains spaces instead of tabs.

Before you start to build the package, change the version of the package to reflect mp3 support. debchange will ask you to add some comments to the changelog.

```
$ debchange -v 1.0-999~mp3+1

darkice (1.0-999~mp3+1) UNRELEASED; urgency=low

  * New build with mp3 support
```

```
 --  <pi@raspberrypi>  Sat, 11 Aug 2012 13:35:06 +0000
```

Build and install the darkice package:

```
$ dpkg-buildpackage -rfakeroot -uc -b
(...)
$ sudo dpkg -i ../darkice_1.0-999~mp3+1_armhf.deb
(...)
Preparing to replace darkice 1.0-999 (using .../darkice_1.0-
999~mp3+1_armhf.deb) ...
Unpacking replacement darkice ...
Setting up darkice (1.0-999~mp3+1) ...
(...)
```

Now you have installed DarkIce with mp3 support.

Configuring Darkice

Add a configuration by copying the template file:

```
$ sudo cp /usr/share/doc/darkice/examples/darkice.cfg /etc/
```

Edit the configuration file:

- Set duration to 0 (record forever), reconnect=yes
- Set the input to the webcam audio in the [input] section
    - device = hw:1,0 # the first number is the card and the second is subdevice found by entering arecord -l
    - sampleRate = 22050
    - bitsPerSample=16
    - channel =1
- Set the icecast stream [icecast2-0]
    - bitrateMode = vbr
    - format = mp3
    - quality = 0.4 # this seems to not draw too much of the CPU
    - server = localhost
    - port = 8000
    - password = whatevertheicecast2serverpasswordis

- mountPoint = webcam
- name = WebcamMic
- description = Web Cam Microphone stream
- url = http://localhost
- genre = audio
- public = no

Starting Darkice on Boot

Add the following in crontab (crontab -e):

@reboot [-x /usr/bin/darkice] && /usr/bin/darkice > /dev/null

Relay Server

To solve the problem of changing IP addresses, I wrote a simple TCP receiver which runs on an AWS server with a fixed IP address. When the TCP receiver receives a message, it will authenticate it and forward it on to the Google messaging service.

Setting access to the TCP socket on AWS:

In the security group used for the EC2 instance, add a new Custom TCP rule which allow the TCP connection to the port

On AWS, install the software, write a script to run the software:

```
#!/bin/bash
# Script to run the relay server for the SA Home Monitor
export PYTHONPATH=/home/ec2-
user/source/HomeMonitorPiServer/HomeMonitorPiServer/src/
export _IP=$(hostname -i)
cd /home/ec2-user/source/HomeMonitorPiServer/HomeMonitorPiServer/src/gcmrelay
python relayserver.py $_IP
```

To make it run on boot, do crontab -e then enter the line @reboot <scriptname> &

Viewing on VLC

The audio and video can be accessed on any PC using the VideoLan VLC player.

Download and install the player from the VideoLan home page.

Access the video stream using the following URL's:

- Video: http://x.x.x.x:80801
- Audio: http://x.x.x.x:8000/webcam