Who's Knocking?

Mark G.

December 8, 2018

This presentation gives an overview of a do-it-yourself, front door, security camera installation. Topics covered are a description of the installation, the camera, the live monitor, and the motion sensing and recording software.

Technologies used in this system are:

- Tools for infrastructure manipulation (ladder, drill, hammer, drywall saw) ;
- Network equipment (power over ethernet switch, VLAN capable switches);
- Security camera;
- Raspberry Pi 3B, running Raspbian, with a seven inch display;
- FreeBSD server running a jailed zoneminder system.

Contents

List of Figures

26 **Touchscreen display parts** (picture attributed to element14 Community) 25

1 Tools and Structure

This section is for completeness. I needed various tools and bits of wood to complete the project.

1. Ladder - I had to both access the attic and reach the soffit. Turns out I needed an 8 foot step ladder and a 10 foot extension ladder. The step ladder was used for mounting the camera to the soffit, while the extension ladder was used to access the attic from inside the garage.
2. Wood - a used a 3/4 inch plank of plywood, and some pieces of two-by-four as a structural base to hold the camera strongly in place. The plywood contacts the top of the soffit from the attic side. It measures about 24 inches by 12 inches so as to fit in between the joists.
3. Drill - I needed to drill a hole in the ceiling of the network closet to run ethernet cables into the attic. I also used it for screwing the camera into the soffit/plywood.
4. Hammer - the plywood was a tight fit and needed some coaxing.
5. Drywall hole saw - this handy saw makes short work of drywall and was used to carve out the mounting hole for the display and its access ports from the network closet side.

1.1 Mounted Camera

The mounted camera is shown in figure 1. The camera is physically mounted on the exterior of the house, through the soffit, and screwed into the 3/4 inch plywood, which is fastened to the interior rafters in the attic.

Figure 1: **Front door camera location overview**

This positioning location allows the camera to have vision of the entire front porch, but does not extend to the street, or any neighbours' property. The camera's field of view is shown in figure 2.

Figure 2: **Camera's field of view**

1.2 Live Display

The live display component was embedded into the living room wall directly adjacent to the network closet. This was a lucky choice, since it removed a large chunk of cabling work. It was the most natural location for the display since it allowed me to interact with the display without also indicating my presence. I made a little movie with some actions scenes; let's see if it works.

Figure 3 shows the embedded display with the display in screen saver mode (blank screen), and thus off.

Figure 3: **Living room wall with live display off**

Figure 4 shows the embedded display with the display on and showing a live stream of the camera.

Figure 4: **Living room wall with live display on**

A look at the inside of the network closet (figure 5) shows the access holes for the RPi and network cable. Figure 6 shows the covered access hole.

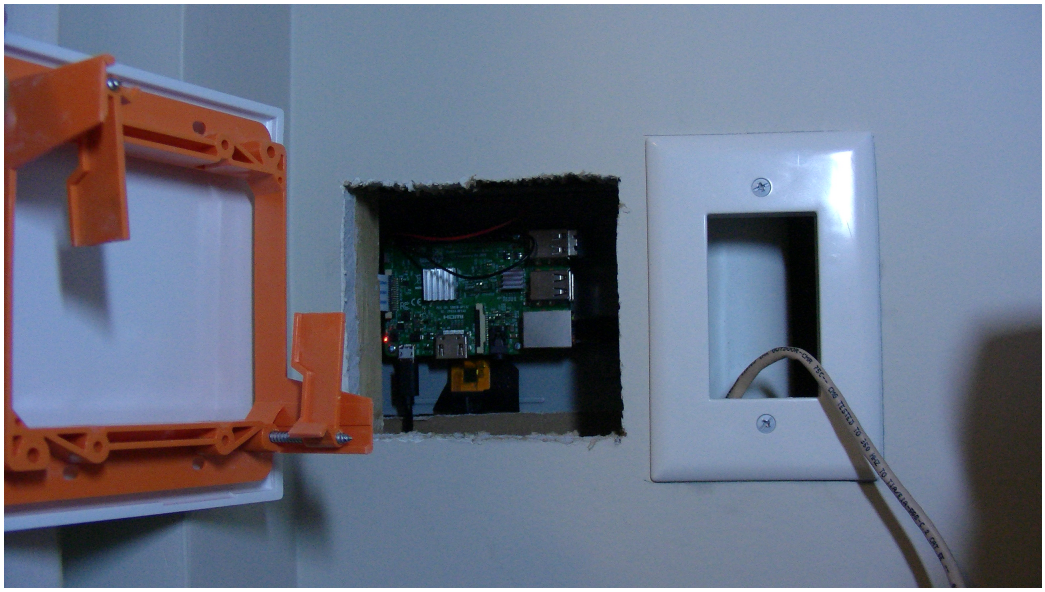

Figure 5: **Network closet access hole to the live display**

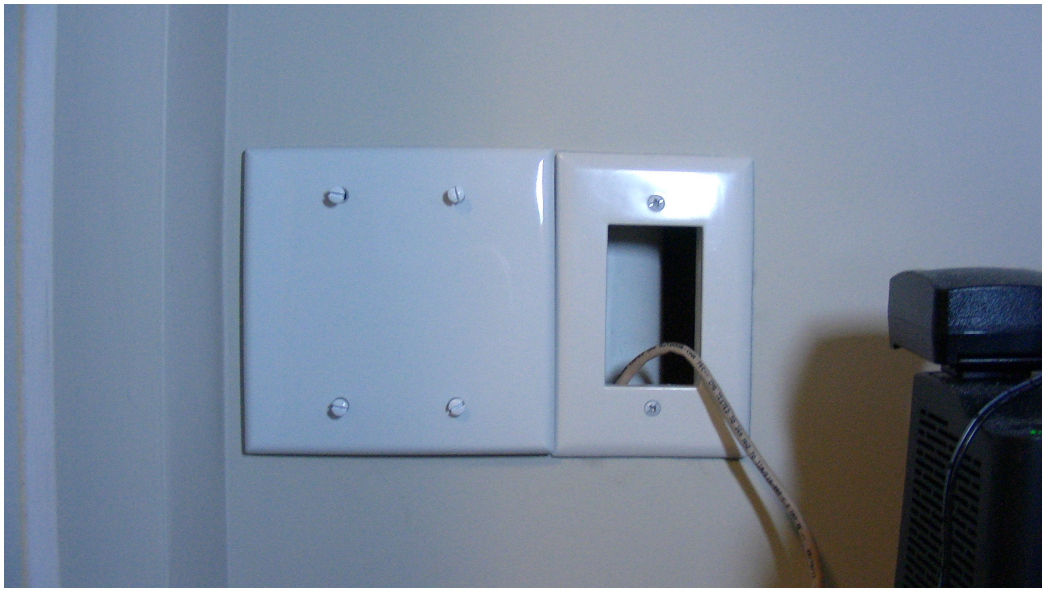

Figure 6: **Network closet covered access hole**

2 Network

The network portion of this project has more depth than this presentation hints at. Some of the issues, briefly, include:

- the IPv4 addressing of the camera, RPi, and FreeBSD server. This address space required creation of a virtual LAN (VLAN) to isolate camera related traffic. By design, neither the camera nor the RPi display computer are accessible from the Internet.
- the IPv6 addressing of the FreeBSD server, which allowed for easy routing to the zoneminder web interface when connecting to my home using my OpenVPN service. This allows secure access to the motion detected videos from outside my home.
- the details of the patch panel installation and its cable connections.
- the details of the power over ethernet (PoE) switch installation. I've given a presentation on PoE and it can be found on the vicpimakers.ca website.

2.1 Network Cables

I ran several category 5e cables into the attic. Figure 7 shows the group of cables running up into the attic from the network closet. From the attic, I sent two cables towards the front door's soffit, another two going to the garage roof and two more to the garage's

workbench wall. I don't have any pictures of the inside of the attic, mainly because I forgot to take them, but they would also be quite uninteresting anyway.

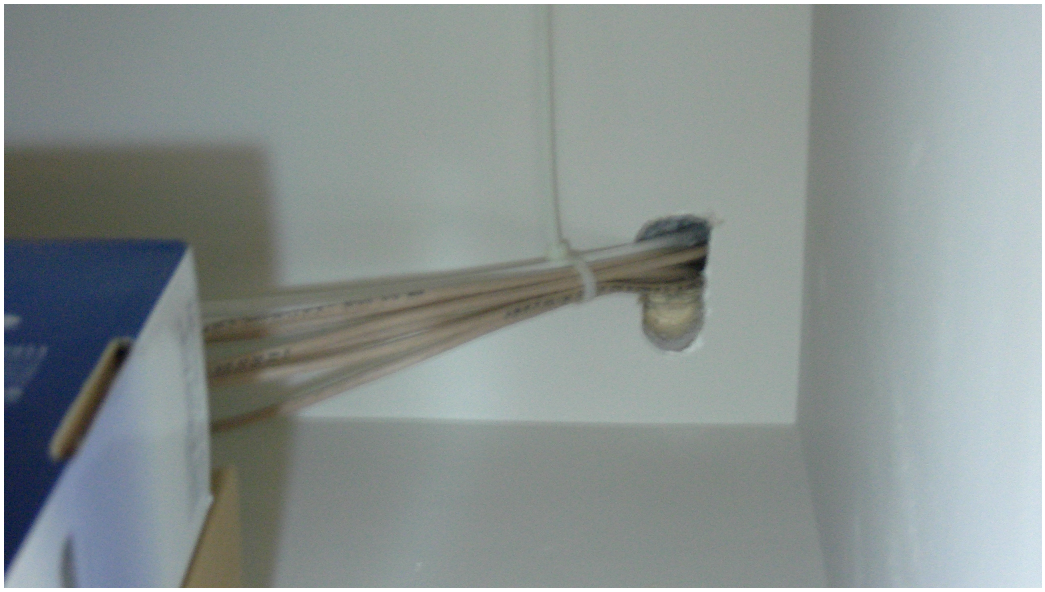

Figure 7: **Network cables running up to the attic**

2.2 Patch Panel

Any network cable installed into a building needs to have the cable endpoints connected to a patch panel. This allows for easy configuration and interconnection with interchangeable, devices, switches or routers. Figure 8 shows a patch panel I installed for this purpose. It is a Trendnet TC-P16C6.

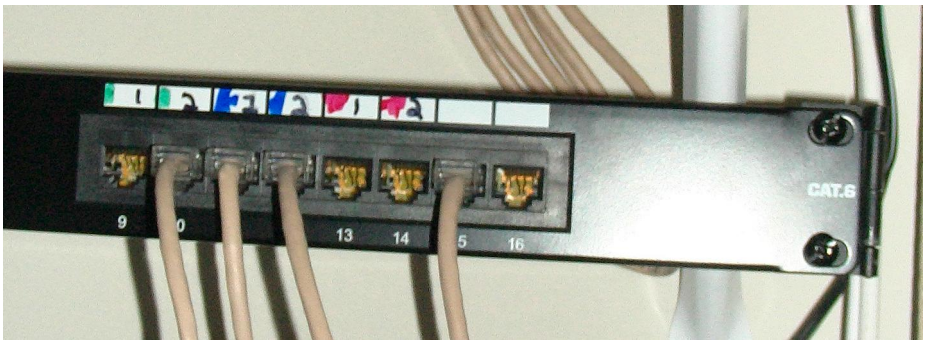

Figure 8: **Patch panel**

2.3 Power Over Ethernet Switch

The camera does not have easy access to power. Adding power circuits requires permits and other expenses, so a solution that doesn't require power upgrades is needed. Power over ethernet turns out to be perfect for this project. This project uses a TP-LINK TL-SG1008PE switch.

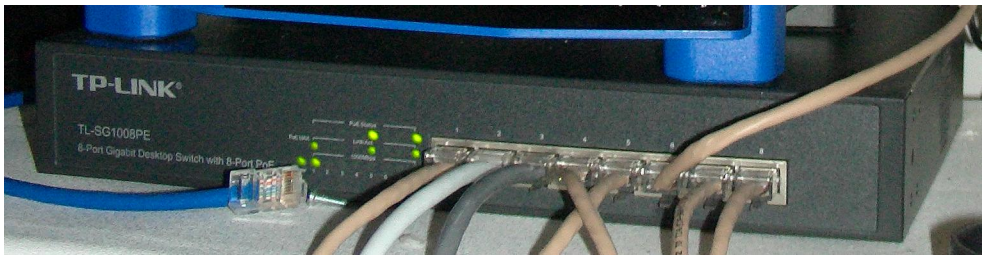

Figure 9: **Power-over-ethernet (PoE) switch**

A slightly more encompassing picture (figure 10) shows the switch and panel together.

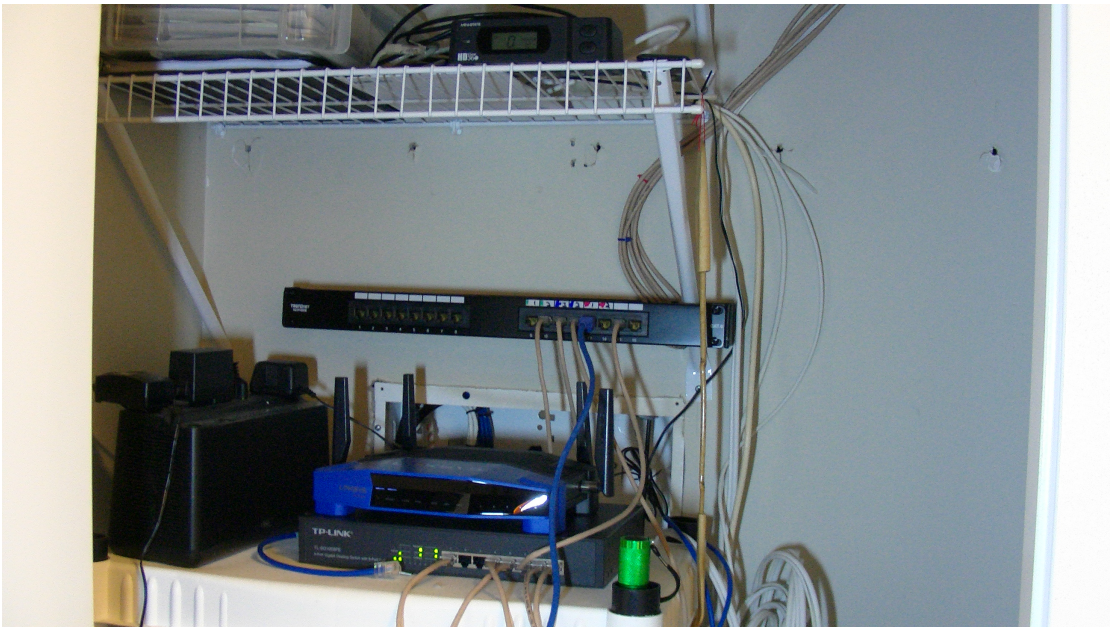

Figure 10: **Fuller view of network closet**

3 Camera

The camera, shown mounted in figure 11, is a Hikvision model DS-2CD2342WD-I (see figure 12 for a view of the camera's retail box). It uses PoE for both power and network connectivity. It is plugged into a jack in the attic which is wired into the patch panel. A short patch cable runs from the camera's patch panel port to the switch.

Figure 11: **Close up of mounted camera**

The advantages of a wired PoE camera are a robust, stable network connection, and no battery issues or high voltage power requirements.

Figure 12: **EXIR Turret Network Camera retail box**

3.1 Configuration

The camera was configured using the Safari web browser on a MacOS X (el capitan) system. A series of image captures of the configuration screens where changes were made, are shown. I won't talk much about the details, they are here for documentation's sake.

3.1.1 Camera System Settings

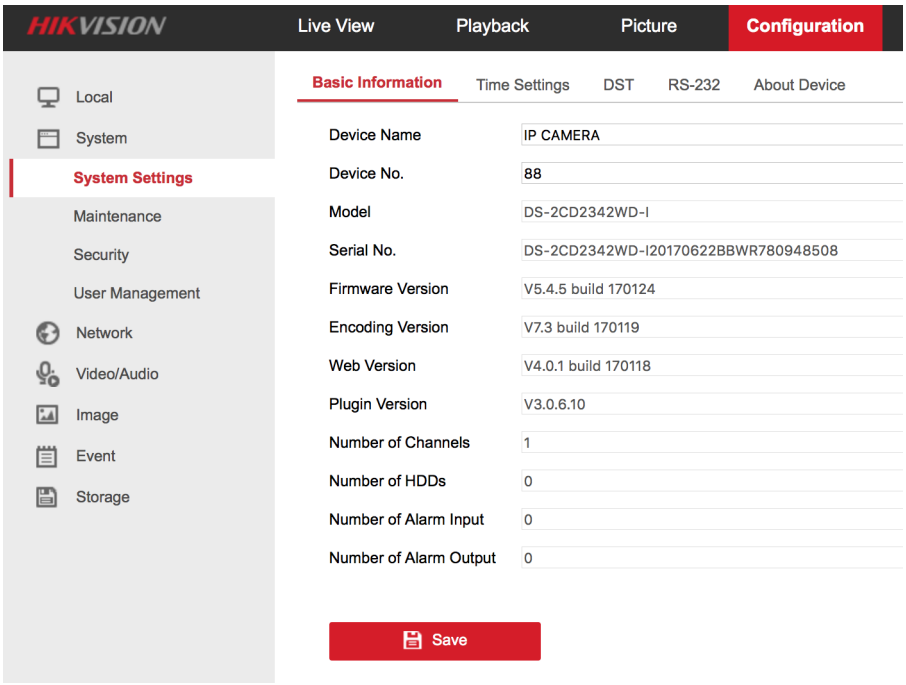

Figure 13: **Basic camera system information for the camera**

The camera uses an internal network NTP server located at 10.9.0.137 (the freebsd zoneminder host). The Raspberry Pi display server is also using this NTP server, so that all the systems involved have their times synchronized.

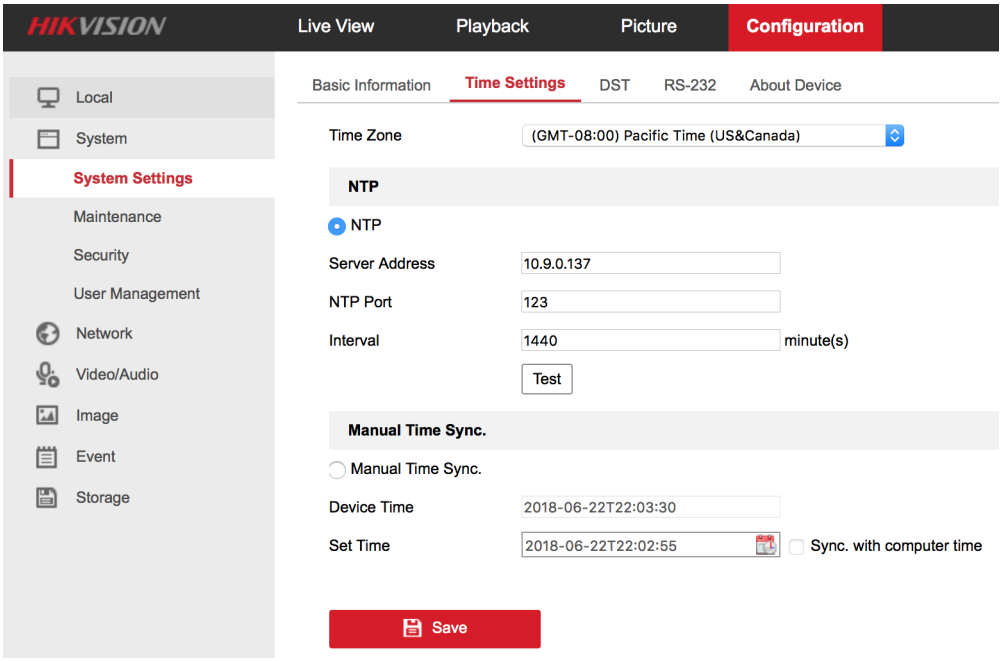

Figure 14: **Camera system time and network time protocol (NTP) settings**

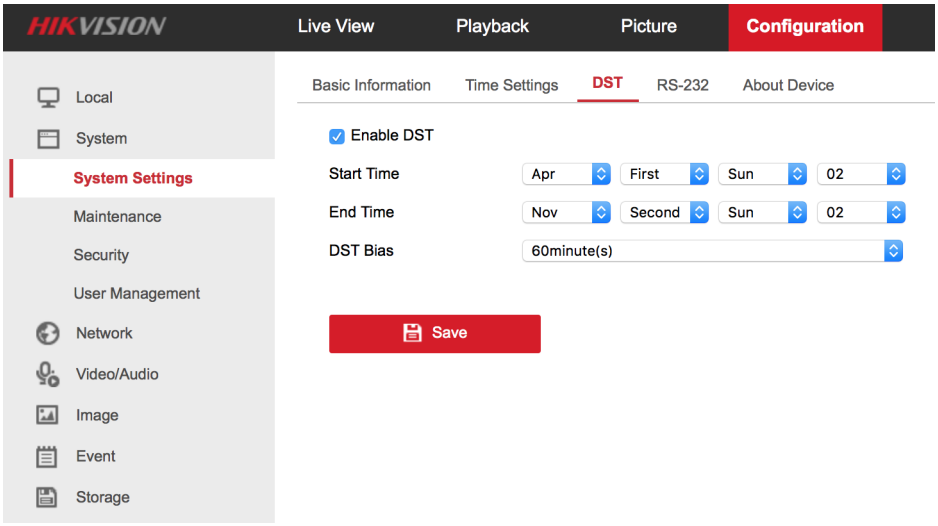

Figure 15: **Camera system daylight savings time (DST) settings**

3.1.2 Camera System Security

The camera uses basic authentication, which is username and password based. I should make sure that the connection is encrypted, but I don't care, since the network is internal only and is low risk. The username and password act to stop casual viewing of the camera's streams.

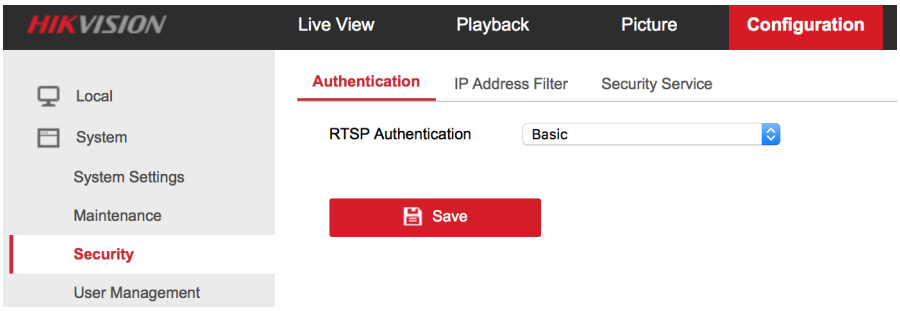

Figure 16: **Camera system security authentication settings**

We'll also restrict the IP addresses that can talk to the camera. The listed ones are the RPi, the FreeBSD server and a small pool of addresses used by miscellaneous other computing devices in my home.

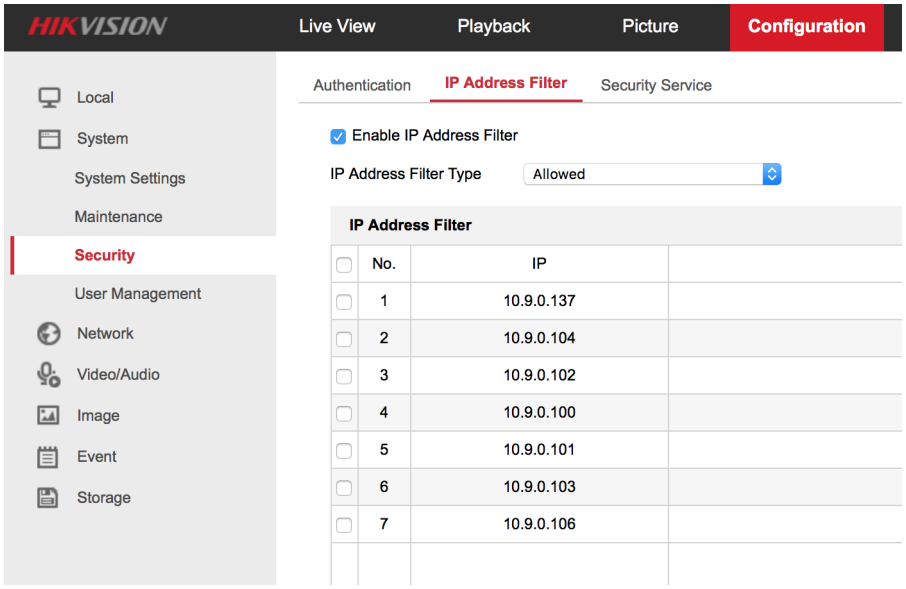

Figure 17: **Camera system security IP address filter settings**

3.1.3 Camera System User Management

The camera allows for the creation of separate users with differing privileges. In the spirit of the best practice of minimizing permissions, I created two more users in addition to the built-in admin user.

- An operator-privileged user named zoneminder was created for the ZoneMinder software. This allows it to be able to read the live feed of the camera and to have some other minor privileges, but does not have admin power.
- An non-privileged user named walleye was created for the RPi display software. This allows it to only be able to read the live feed of the camera. It does not have any admin power.

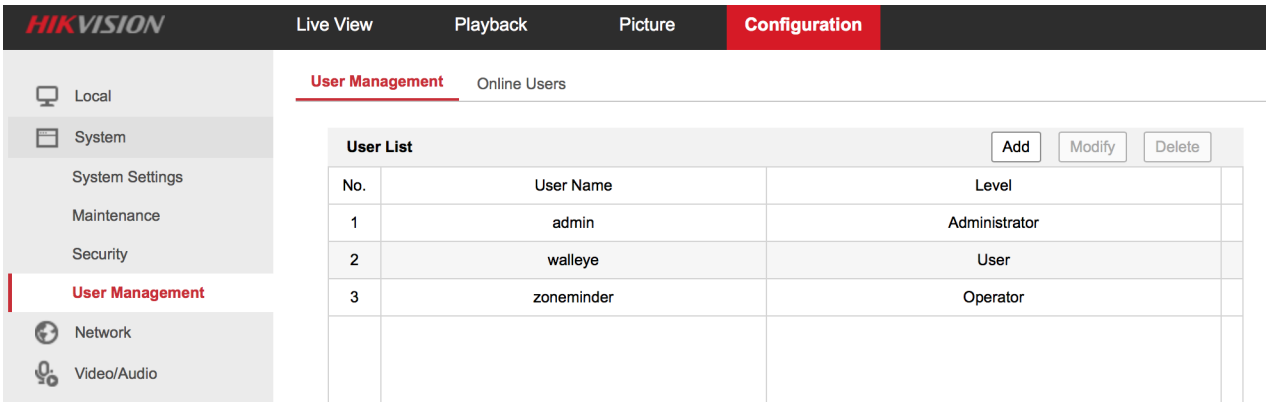

Figure 18: **List of camera users showing walleye display user**

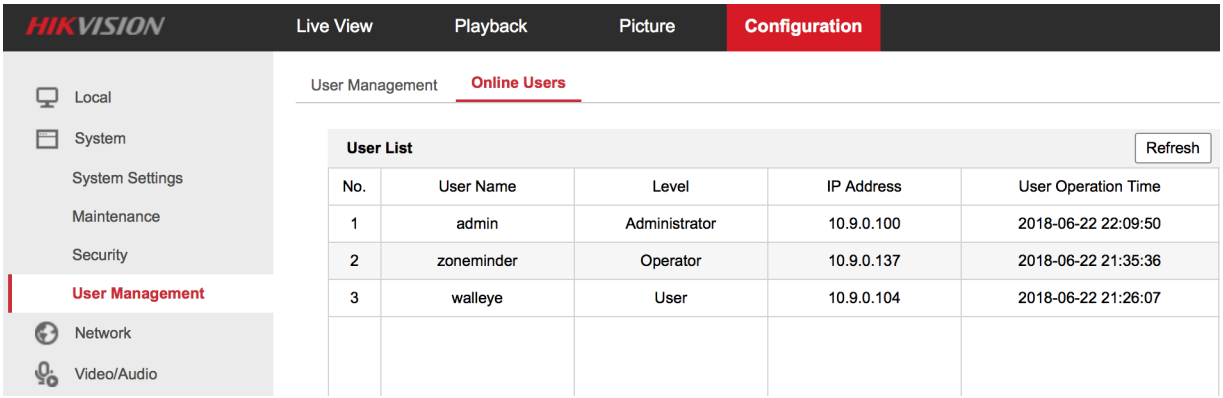

Figure 19: **List of online camera users**

3.1.4 Camera Network Basic Settings

Currently the camera has a static IPv4 address of 10.9.0.88. It has no default gateway, so as to limit, hopefully, any chatter to other systems on the network. I have not done the work to prove this assertion.

I won't be giving it an IPv6 address, since I want to severely limit its communication options.

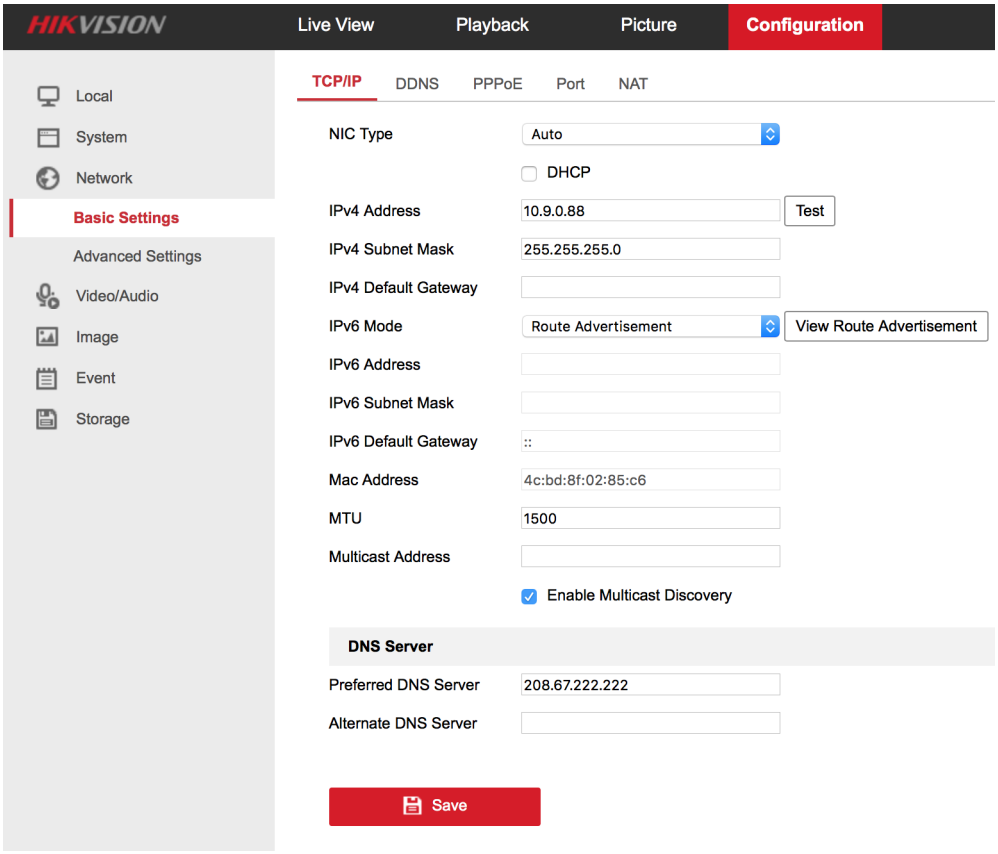

Figure 20: **Camera network, basic IP settings**

It is worthwhile to make a note of the ports involved in the camera's communication protocols. There seem to be higher numbered UDP ports involved in some parts of the streaming behaviour of an RTSP connection. Some investigation of the RTP/RTSP mechanisms is warranted.

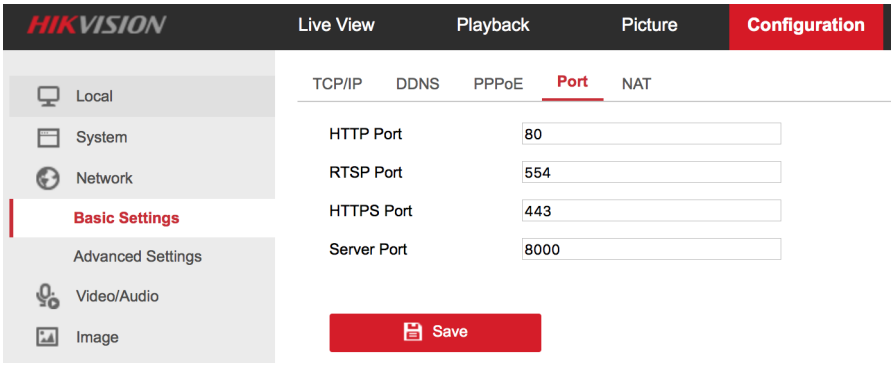

Figure 21: **Camera network, basic port settings**

3.1.5 Camera Network Advanced Settings

There are a number of network protocols supported by the camera. I don't need most of them, so it is sensible to turn them off. Below, we disable the simple network management protocol (SNMP) as an example.

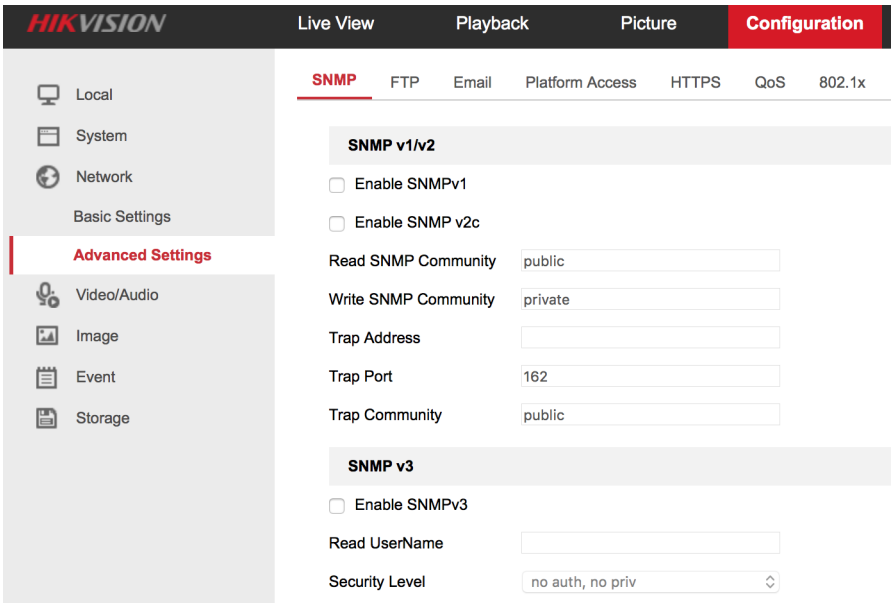

Figure 22: **Camera network, advanced SNMP disabled settings**

Under the HTTPS tab, I have enabled TLS, but the installed certificate is likely to become useless at some point. I am unsure if I can change it very easily to one which I sign with my own certificate authority.

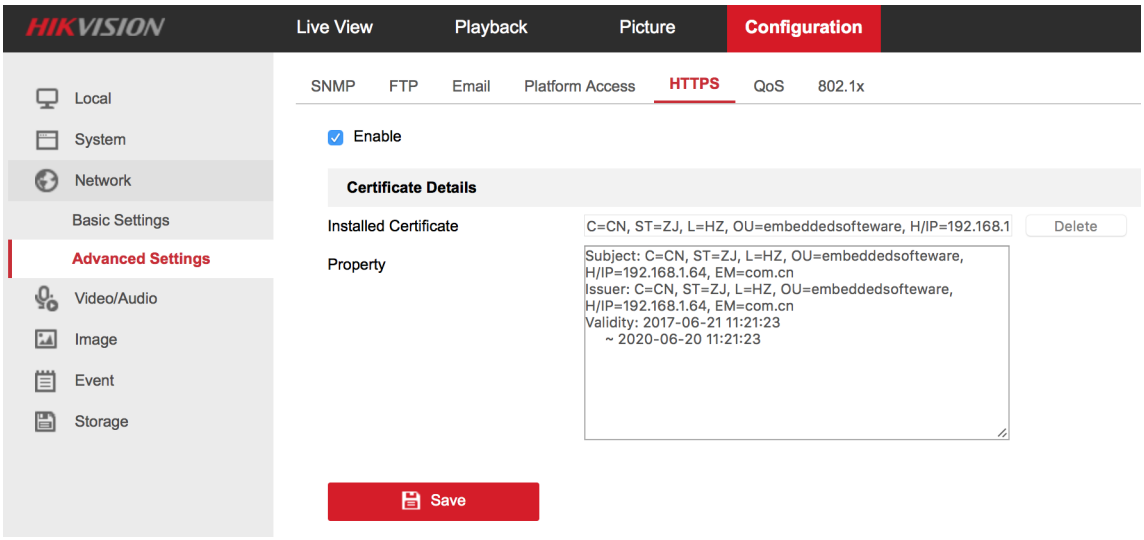

Figure 23: **Camera network, advanced HTTPS settings**

3.1.6 Camera Video / Audio Settings

Most of these are the defaults and I have only a cursory knowledge of what they mean/do.

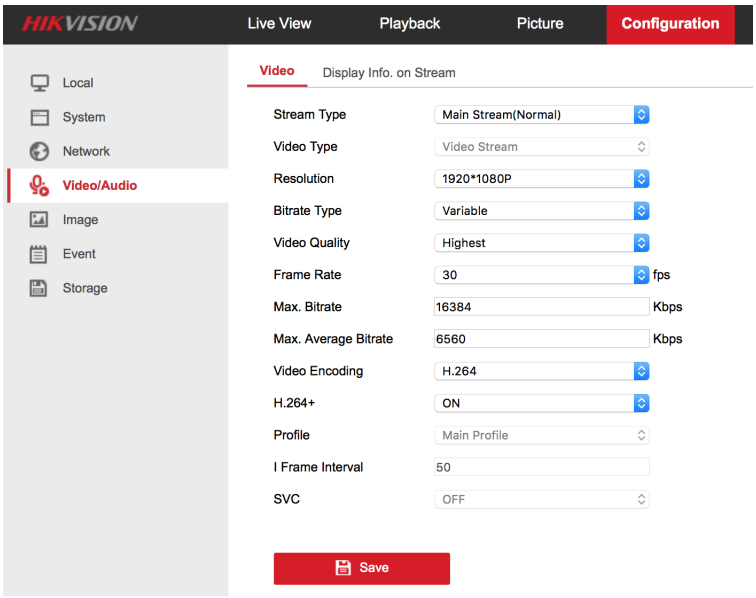

Figure 24: **Camera Video settings**

4 Raspberry Pi Live Display

The live display system uses an element14 brand, 7-inch, touchscreen display designed specifically for the Raspberry Pi. It is compatible with models 3B, 2B and others. The latest version of Raspbian works perfectly with the display.

A simple wireless keyboard is used for any typing needed, such as starting the display software.

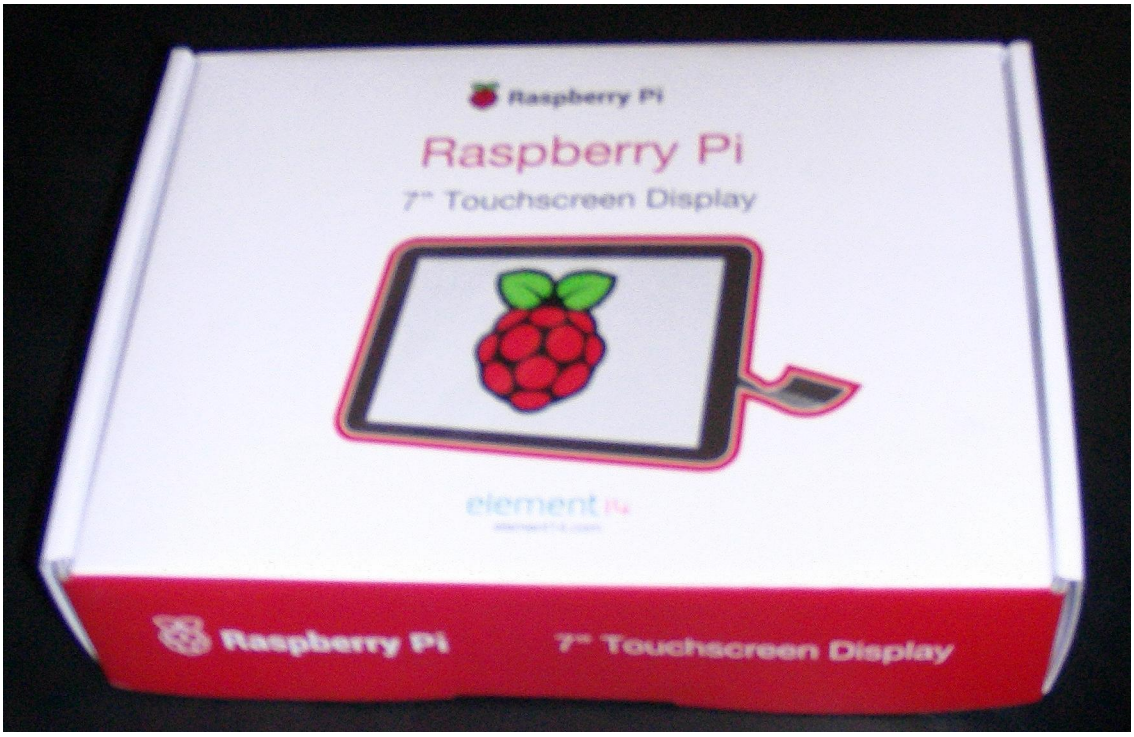

Figure 25: **Element14 raspberry pi touchscreen display**

Figure 26 shows what's in the box. This illustration was borrowed from the element14 documentation.

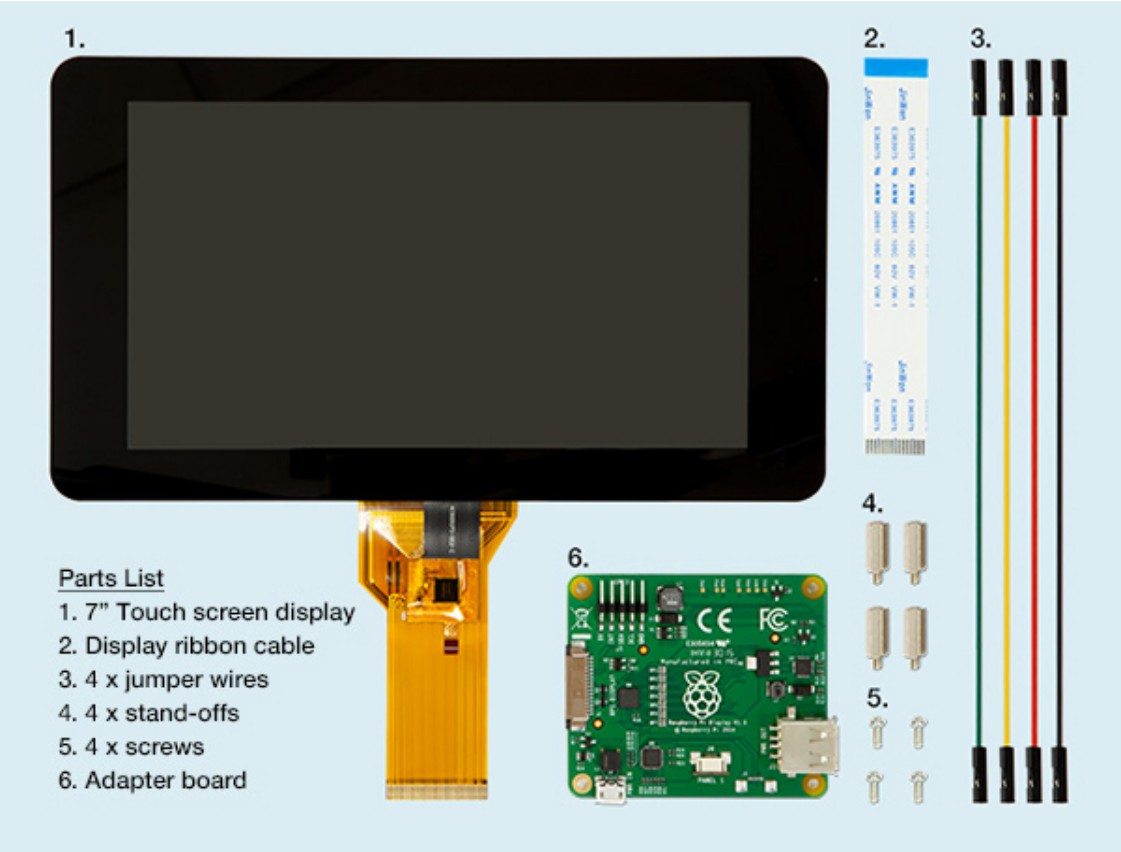

Figure 26: **Touchscreen display parts** (picture attributed to element14 Community)

4.1 Process and Method for Use

The Pi and display combination is intended to be single purpose. It functions as a display only, but may be used as an MQTT broker in the future.

The overall workflow / method of use is as follows:

- automatically boot into the Raspbian default desktop;
- manually start a terminal window;
- run, within that terminal, a loop-control shell script, which starts the software that reads the camera's stream and places it on the display.

Once the loop-control script starts, the display is overlaid with the omxplayer's output, which is the camera's video feed. The entire display is used. Some important notes about operating procedure:

- The screen saver function of the desktop environment must be set to blank screen (basically, turn off the display's backlight).
- The loop script must be started and the camera display is then active.
- When the screen saver kicks in and turns off the display, the live camera feed is no longer visible and the screen is blank.
- To re-enable the display, a simple tap on the screen will disengage the screen saver and the live display is shown. I have a small video of this action.
- The screen saver can also be disengaged by tapping a key on the wireless keyboard, as well clicking the wireless mouse's button.

After experimentation, it turns out that memory leaks are a concern if we run the omxplayer for too long a time. This is why we created a loop-control script. The result of running a loop script is that our live display is restarted every 10 minutes (can be adjusted), and we avoid the eventual, observed laggy behaviour when running the program for a long time. This lag is evidenced by a build up of UDP packets on the program's port queue (seen using the netstat command).

4.2 Configuration

The RPi was configured according to the display's documentation found at element14's website [1]. The hardware is shown in figure 26. It is useful to switch over to the website briefly and review the installation instructions.

Currently the Pi has an IPv4 address of 10.9.0.104 given by a DHCP server. Arguably, this should be changed to a static value, but the likely hood of another address being

[1] https://www.element14.com/community/docs/DOC-78156/l/raspberry-pi-7-touchscreen-display

assigned is very low, plus we allow a few of the DHCP address pool to access the camera via an IP filter.

The Pi uses an internal NTP server located at 10.9.0.137 (the freebsd zoneminder host). The camera is also using this NTP server, so that all the systems involved have their times synchronized.

4.3 OMXPlayer

The omxplayer software is used to fetch and display the camera's real-time streaming protocol's (RTSP) feed. This video player was optimized for the raspberry pi and is very fast in regards to video display.

Figuring out how to control the omxplayer was a challenge. It is designed to respond to key presses sent via standard input (stdin), which is usually the keyboard.

Just leaving it running turned out to be problematic. Eventually, the software starts queuing the incoming UDP packets and no longer properly displays the camera view. The only fix to this is to regularly stop and restart the program.

Stopping the player requires having the letter 'q' sent to the stdin stream. Normally this is done by hitting q on the keyboard. This is useless for automation. The Internet provided the answer, which is to redirect omxplayer's stdin to a first-in first-out (FIFO) pipe file.

The creation of the fifo was done as follows, in the /home/pi/ folder:

```
pi@walleye ~: mkfifo omxcontrol
```

Since omxplayer can be told to listen for commands from the fifo file, key presses can be sent to the fifo, such as 'q' or 'p', to control it. Sending a keypress is simple with the 'echo' program, and is shown in the scripts below.

A number of small shell scripts were created to control the display of the video feed. All scripts are in a folder named /home/pi/omxbuttons/. This was the folder name I chose when I was first started trying to figure out how to use omxplayer. There aren't any actual buttons.

4.3.1 loop_control.sh

The loop-control script which runs in a loop forever.

```
#!/bin/sh
# Loop to stop, start omxplayer

OMX_HOME=/home/pi/omxbuttons

while true

do
    # Create an instance of omxplayer listening to the stream
    $OMX_HOME/omx.sh
    sleep 10

    # Start it - basically, press the play key sequence
    $OMX_HOME/start.sh

    # Let it run for 10 minutes and the restart
    sleep 600
    echo "Stopping..."

    # Stop it - send the quit key sequence
    $OMX_HOME/q.sh
    sleep 5
    echo "Restarting..."

    # return to the top of the loop and recreate the omxplayer instance

done
```

Each of the three shell scripts called by the loop script have the following contents.

4.3.2 omx.sh

This script starts the omxplayer program with the required command line arguments.

```
#!/bin/sh
omxplayer --win "0 0 800 480"  rtsp://walleye:password@10.9.0.88
 < /home/pi/omxcontrol &
```

The --win option is used to place the image over the entire screen, starting at x position 0 (left) and extending 800 pixels (width), and y position 0 (top) extending 480 pixels (height).

The next argument is the RTSP URL that determines the stream to be read. As discussed earlier, a low-privilege user named walleye was created for the RPi to have just a live feed of the camera and to have no other privileges (see figure 18). This user is seen in the streaming source parameter (rtsp URL). This means that the omx.sh file should have restricted permissions to keep the password protected.

The command line clause " < /home/pi/omxcontrol " tells omxplayer to substitute the named fifo as standard input, which means it will listen for key presses on that file-like device. Characters that are written to the fifo will be read as key presses by the omxplayer. This provides a means for automated control.

We put the omxplayer command in the background (as seen by the use of the ampersand, &, at the end of the line), so that it continues to run after the omx.sh script exits, and returns control to the loop script.

4.3.3 start.sh

This script sends a set of play command characters, 1 and 2, to the omxcontrol fifo, which are picked up by the omxplayer executable, which then begins the live display of the stream.

```
#!/bin/sh
echo -n 12 > /home/pi/omxcontrol
```

The -n argument tells echo not to include a newline in its output.

4.3.4 q.sh

This script sends a quit command character, q, to the omxcontrol fifo, which is picked up by the omxplayer executable, which then exits.

```
#!/bin/sh
echo -n q > /home/pi/omxcontrol
```

5 ZoneMinder - FreeBSD Jail on VLAN

This section won't really be covered unless there is time / interest. I'll simply show the zoneminder console and some captured videos.

The zoneminder software is dependency heavy and has many network services. It will therefore be put into its own jail.

5.1 Jail Network Settings

We will use the VLAN 9 interface (vr0.9) as the jail's network interface, with an IPv4 address of 10.9.0.80, effectively adding it to VLAN 9.

5.2 Host /etc/jail.conf Section

The host system's /etc/jail.conf file is updated to add the following zoneminder jail configuration section:

```
src {
    path = /usr/home/jail/sourcecontrol;
    mount.devfs;
    host.hostname = src.palaceofretention.ca;
    host.domainname = palaceofretention.ca;
    ip4.addr = vr0.9|10.9.0.80;
    ip4.addr += lo1|127.0.80.1;
    ip6.addr = re0|fdea:667d:9747:1012::80;
    exec.start = "/bin/sh /etc/rc";
    exec.stop = "/bin/sh /etc/rc.shutdown";
}
```

Observe the network addresses:

```
root@nas:/usr/jails/zoneminder/etc # jail -c zoneminder
zoneminder: created
ELF ldconfig path: /lib /usr/lib /usr/lib/compat /usr/local/lib
32-bit compatibility ldconfig path: /usr/lib32
Setting hostname: zoneminder.palaceofretention.ca.
Creating and/or trimming log files.
Starting syslogd.
Clearing /tmp.
Updating motd:.
Starting cron.

Wed Oct 17 02:59:30 UTC 2018
```

```
root@nas:/usr/jails/zoneminder/etc # jexec zoneminder /bin/tcsh
root@zoneminder:/ # ifconfig
...
re0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=8209b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM,
        WOL_MAGIC,LINKSTATE>
        ether 00:13:4b:0f:d6:b5
        hwaddr 00:13:4b:0f:d6:b5
        inet6 fdea:667d:9747:1012::80 prefixlen 128
        nd6 options=23<PERFORMNUD,ACCEPT_RTADV,AUTO_LINKLOCAL>
        media: Ethernet autoselect (1000baseT <full-duplex>)
        status: active

vr0.9: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=80000<LINKSTATE>
        ether 00:11:94:d8:83:ae
        inet 10.9.0.80 netmask 0xffffffff broadcast 10.9.0.80
        nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
        media: Ethernet autoselect (100baseTX <full-duplex>)
        status: active
        vlan: 9 vlanpcp: 0 parent interface: vr0
        groups: vlan
```

5.3 Jail /etc/rc.conf

The jail's system rc file is required to signal what services are enabled in the jail.

```
# Keep tmp tidy by emptying on startup
clear_tmp_enable="YES"

# do not open any network sockets for syslgod
syslogd_flags="-ss"

# leave sendmail off
sendmail_enable="NONE"

# do not start the DNS caching server
local_unbound_enable="NO"

hostname="zoneminder.palaceofretention.ca"

# zoneminder related services
nginx_enable="YES"
```

```
php_fpm_enable="YES"

# Required for zoneminder (also uses nginx)
mysql_enable="YES"
mysql_server_enable="YES"
mysql_dbdir="/var/db/mysql"

fcgiwrap_enable="YES"
fcgiwrap_user="www"
fcgiwrap_socket_owner="www"
fcgiwrap_flags="-c 4"
zoneminder_enable="YES"
```

5.4 ZoneMinder Jail Installation

This section documents installing the zoneminder system into the jail. We had a previous non-jailed instance of zoneminder and we can copy over the mysql databases for reuse.

Now that we have an upgraded jail (as of FreeBSD 11.2-p4), we'll use packages:

```
root@zoneminder:/usr/local/www # pkg install zoneminder-h264
Updating FreeBSD repository catalogue...
FreeBSD repository is up to date.
All repositories are up to date.
The following 132 package(s) will be affected (of 0 checked):

Proceed with this action? [y/N]: y
...
Message from zoneminder-h264-1.30.20170222_7:

ZoneMinder is a free, open source Closed-circuit television software
application developed for Unix-like operating systems which supports
IP, USB and Analog cameras.

New installs
============

ZoneMinder requires a MySQL (or MySQL forks) database backend and
a http server, capable to execute PHP and CGI scripts.

To simplify things, we assume, that you use MySQL and NGINX on
the same server.

1. Preliminary steps
```

1.1 Install databases/mysql56-server or newer
You may choose your favourite method - ports or packages here.
FreeBSD default setting use STRICT_TRANS_TABLES sql_mode.
It's mandatory to disable it. Edit your my.cnf accordingly

The following SQL mode should be compatible with ZM:
sql_mode= NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,
NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION

ZoneMinder use very simple queries, however it tends to write to
the database quite a lot depending on your capture mode and number
of cameras. So tweak your MySQL instance accordantly

Now, enable and start MySQL
sysrc mysql_server_enable="YES"
service mysql-server start

1.2 Install www/nginx
We provide an example for an HTTP install, however, you should use
HTTPS if you plan to expose your installation to the public. There
are plenty guides how to do it and security/letsencrypt.sh is a
good way to get a valid SSL certificate.

Your server block should include the following:

```
server {
    root /usr/local/www/zoneminder;
    try_files $uri $uri/ /index.php$is_args$args;
    index index.php;

    location = /cgi-bin/nph-zms {
        include fastcgi_params;
        fastcgi_param   SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_pass    unix:/var/run/fcgiwrap/fcgiwrap.sock;
    }

    location ~ \.php$ {
        include fastcgi_params;
        fastcgi_param   SCRIPT_FILENAME  $document_root$fastcgi_script_name;
        fastcgi_pass    unix:/var/run/php-fpm.sock;
    }

    location /api {
```

```
            rewrite ^/api/(.+)$ /api/index.php?p=$1 last;
        }
    }
```

1.2.1 ZoneMinder has it's own authentication system, however it's recommend to use NGINX basic auth over HTTPS if you don't need fine grain control to ZoneMinder components.

1.2.2 If you choose ZoneMinder authentication, it's recommended to prohibit access to image and events folder as it's possible to guess file names inside it.

```
    location ~ ^/(?:images|events)/ {
            deny all;
    }
```

Enable and start NGINX

```
    sysrc nginx_enable="YES"
    service nginx start
```

1.3 Install www/fcgiwrap

As NGINX lacks it's own CGI wrapper, we need external one. Please note that ZoneMinder's montage page use simultaneous access to all cameras, so you need to use at least as many fcgiwrap workers as your number of cameras. The following example assumes you have 4.

Enable and start FcgiWrap

```
    sysrc fcgiwrap_enable="YES"
    sysrc fcgiwrap_user="www"
    sysrc fcgiwrap_socket_owner="www"
    sysrc fcgiwrap_flags="-c 4"
```

1.4 PHP is installed as a dependency to ZoneMinder. However, you should tweak some of it's settings.

Edit /usr/local/etc/php-fpm.conf and set

```
    listen = /var/run/php-fpm.sock
    listen.owner = www
    listen.group = www
    env[PATH] = /usr/local/bin:/usr/bin:/bin
```

If you want to set another path for the socket file, make sure you change it in your NGINX config well. The env[PATH] needs to be set to locate the zip utility as ZoneMinder's export functions rely on

```
    exec(). Sorry, chroot folks.

    PHP throws warning if date.timezone option is not set. The best place
    to do it is to create new ini file in /usr/local/etc/php with overrides

        date.timezone = "UTC"

    Enable and start php-fpm
        sysrc php_fpm_enable="YES"
        service php-fpm start

1.5 ZoneMinder constantly keeps the last N frames from its cameras to
    preserve them when alarm occurs. This can be a performance hog if
    placed on spindle drive. The best practice is put it on tmpfs.
    See https://www.freebsd.org/cgi/man.cgi?query=tmpfs for more
    information.

    ZoneMinder will use /tmp for default. If you plan to change it, see
    ZM_PATH_MAP setting.

    Mapping /tmp to tmpfs is actually a recommended step under FreeBSD.
    Edit /etc/fstab and add the following:

        tmpfs            /tmp        tmpfs    rw,nosuid,mode=0177700

    The size of temporary files depends on your number of cameras
    number and frames you plan to keep. My 12 3Mbit cameras with 25
    last frames consumes 6 GB.

2. ZoneMinder installation

    Connect to MySQL under root and create zm user and populate database.

    mysql -u root -p

        CREATE DATABASE zm;
        GRANT ALL PRIVILEGES ON zm.* TO 'zmuser'@'localhost' IDENTIFIED BY 'zmpass';
        FLUSH PRIVILEGES;
        quit;

    mysql -u root -p zm < /usr/local/share/zoneminder/db/zm_create.sql

2.1 If you have chosen to change the ZoneMinder MySQL credentials to something
    other than zmuser/zmpass then you must now edit /usr/local/etc/zm.conf. Change
```

```
    ZM_DB_USER and ZM_DB_PASS to the values you created in the previous step.

    Enable and start ZoneMinder
        sysrc zoneminder_enable="YES"
        service zoneminder start

Upgrades
========

1. Stop ZoneMinder
    service zoneminder stop

2. Upgrade database
    sudo -u www zmupdate.pl

3. Start ZoneMinder
    service zoneminder start
```