Abusing YouTube for transit and storage.

Rob Keizer - March 2018

Brief Overview

Socks Proxy | qrencode | ffmpeg | YouTube

- Found a socks v5 compatible node (javascript) server
- For every ‘chunk’ of data off the stream, spawn `qrencode | convert`, read the contents from stdout.
- node script listens on a particular port, anything that connects to it, spews data.

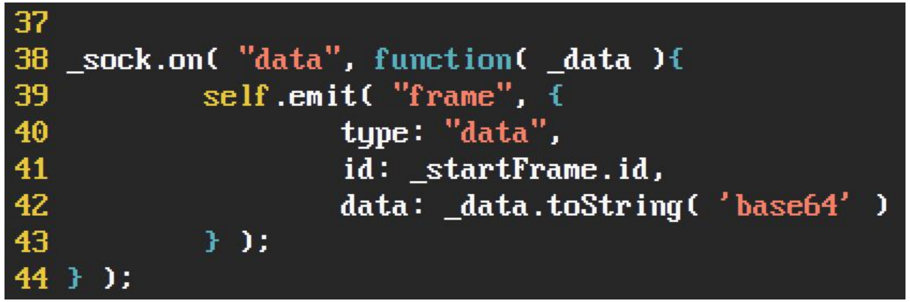
- nc read from particular port, pipe into ffmpeg.
- ffmpeg **image2pipe** read images and create video stream.
- send the output to a flv endpoint for youtube live.

YouTube Live Details

- RTMP Streaming
- 60 FPS
- Use a constant bit rate
- Max resolution of 4k (3840x2160)
- ~50Mbit maximum stream size

Socks Proxy

- Low hanging fruit to get data in.
- Javascript implementation for ease of hackability.
- Create a module around another existing module, using event emitters to get data out.

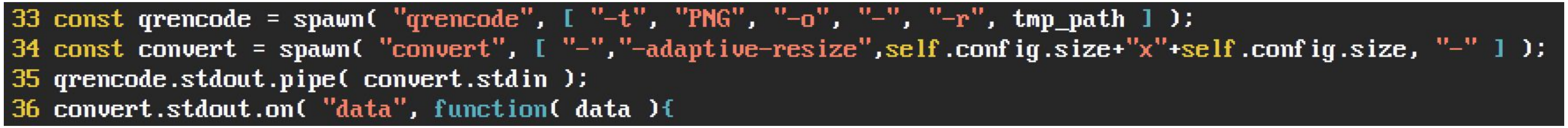


```

_sock.on( "data", function( _data ){
        self.emit( "frame", {
                type: "data",
                id: _startFrame.id,
                data: _data.toString( 'base64' )
        } );
} );
```

encode

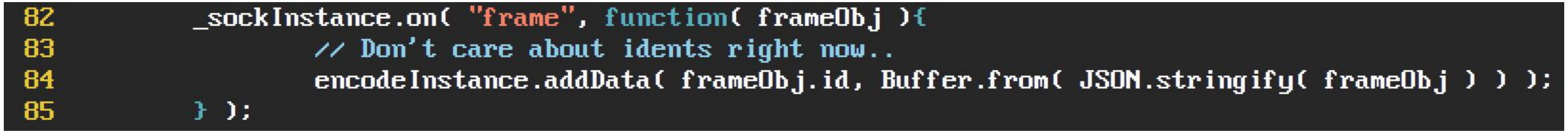
- Takes in an identifier and some data.
- Writes it to a temporary location
  - Thanks qrencode.
  - Workaround most likely possible with some hacking (named pipes, etc).
- Setup to either run locally or hit a web server.
- Uses events and streams in javascript.
- *“qrencode -t PNG -o - -r /tmp/some_file | convert - -adaptive-resize 480x480 -”*



```
const qrencode = spawn( "qrencode", [ "-t", "PNG", "-o", "-", "-r", tmp_path ] );
const convert = spawn( "convert", [ "-","-adaptive-resize",self.config.size+"x"+self.config.size, "-" ] );
qrencode.stdout.pipe( convert.stdin );
convert.stdout.on( "data", function( data ){
```

run.js

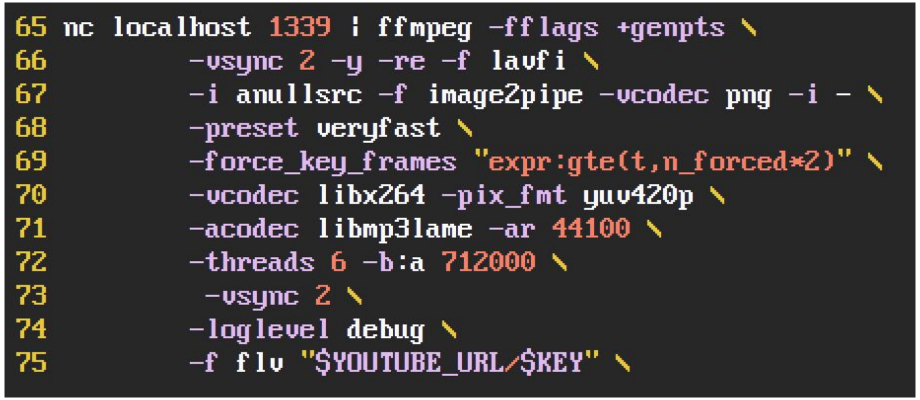
- Glue between socks proxy, encoder.
- Listens on tcp port, sends data to anything that connects.
- Has a loop to ensure any clients get a steady stream of data.
- Creates instance of sock server
- (almost) everything is event driven



```
        _sockInstance.on( "frame", function( frameObj ){
                // Don't care about idents right now..
                encodeInstance.addData( frameObj.id, Buffer.from( JSON.stringify( frameObj ) ) );
        } );
```

ffmpeg

- netcat grabs data from node
- image2pipe takes png’s and converts them to a frame
- output heads to a flv endpoint for youtube live.
- genpts ensures good timestamps on the video stream.
- vsync forces frames to have timestamp
- anullsrc is awesome
- youtube likes keyframes every 2 seconds.



```
nc localhost 1339 | ffmpeg -fflags +genpts \
        -vsync 2 -y -re -f lavfi \
        -i anullsrc -f image2pipe -vcodec png -i - \
        -preset veryfast \
        -force_key_frames "expr:gte(t,n_forced*2)" \
        -vcodec libx264 -pix_fmt yuv420p \
        -acodec libmp3lame -ar 44100 \
        -threads 6 -b:a 712000 \
         -vsync 2 \
        -loglevel debug \
        -f flv "$YOUTUBE_URL/$KEY" \
```

Gotchas

- Youtube doesn’t have great debug.
    - Pass audio, otherwise it ‘works’, but doesn’t really.
- Keep a log of what works/didn’t when it comes to commands.
    - ffmpeg arguments number > 20 for ideal.
- Forgo named pipes in favour of tcp sockets, things tend to handle it better.
    - `tail -f` piped to mplayer for example, works half the time.

Going Forward

- FUSE filesystem
- IP over YouTube
  - Requires 2 live streams
- Steganography
  - Good luck detecting it
- Audio channel
- Error correction
  - Resolution
  - Dropped frames
  - Decoding speeds
- HCC2D (adding colour dimensions)

Questions? Ideas?

- robert@keizer.ca
- @robertkeizer_