How to build better more secure KVM with off-the shelf hardware

An experiment in security-by-design

Tim Panton (tim@pi.pe) (CTO pi.pe gmbh)

Who

Tim Panton: CTO pi.pe GmbH

- Internet Security scanning service 20 years ago
- VoIP 15 years ago
- IoT secure stack for cameras - last 4 years

Goal

Remote access to server assets

- Like the screen/kbd trolley you used to wheel around a DC
- But on the internet
- Plug hardware into hdmi + usb to get access
- Manage a firewall appliance - but not through it!
- Cheap low power arm servers don't do LOM
- Easy to use

Threat model

What are we protecting?
What are we assuming?
Protecting from whom?

- Protect Service (don’t provide new attack vectors)
- Protect Auth (don’t allow collection of passwords etc)
- Protect Data (don’t allow interception in flight)
- Assume ‘lazy but not evil’ users
- Assume untrusted infra structure
- Assume ‘semi secure’ location - e.g. DC rack with CCTV
- Main threat is external (over Network) from:
    - Automated scans
    - Active targeting by motivated individuals
      (assymentry: $1k buys a lot of motivation)

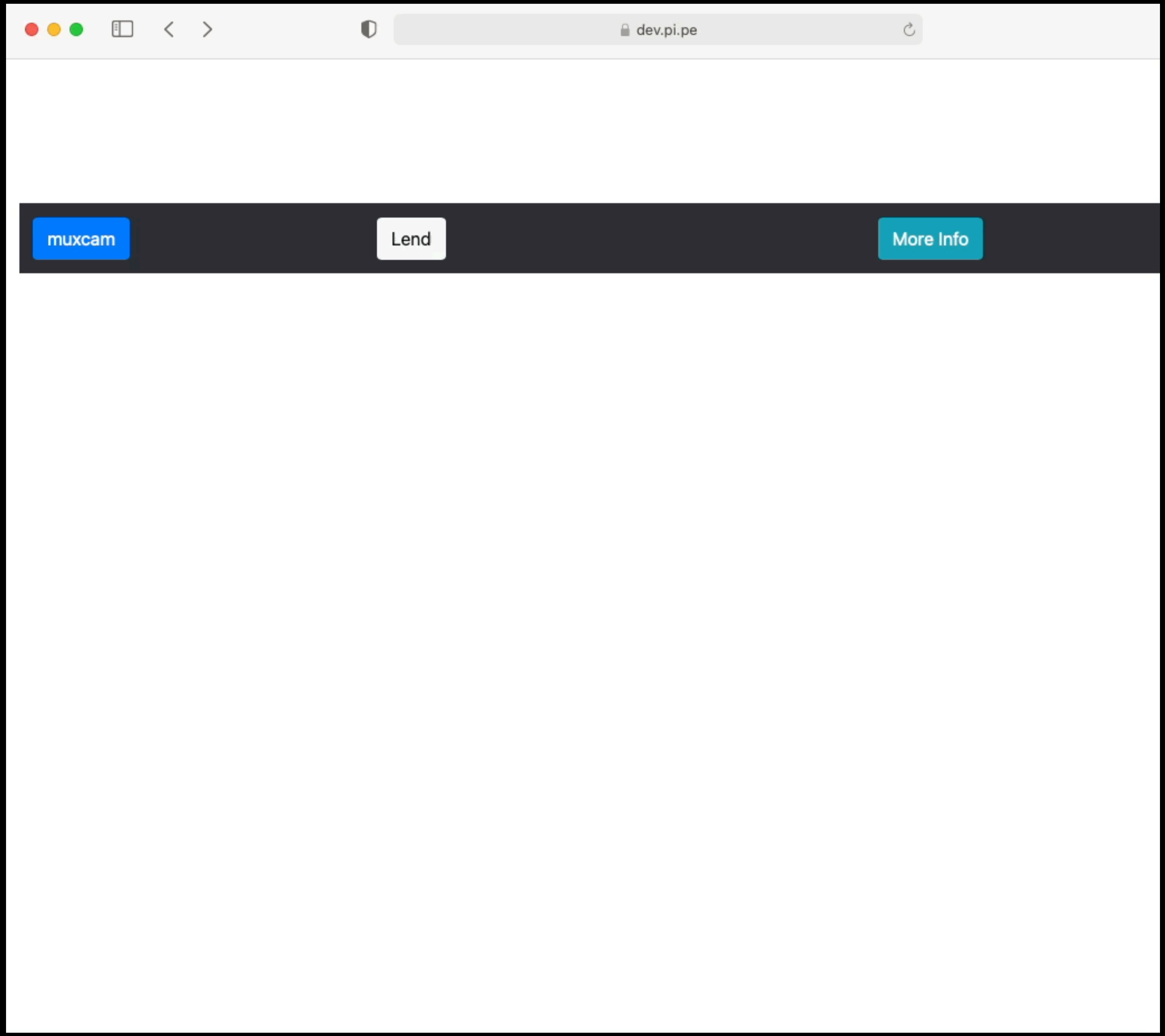
Threat minimization

Three intersecting strategies

- Block known attack vectors (and adjacent)
  - Buffer overun
  - Type trickery
  - Input validation
- Minimize attack surface
  - Simplify interfaces
  - Reduce optionality
  - Minimize secret data usage
- Best practice
  - Leverage standards
  - Tooling, code etc

Result
(Video)

- 1024x768 @30
- From hdmi Ubuntu x86
- To M1 Mac safari
- Via Raspi 4
  capturing hdmi
  and emulating USB mouse + kbd
- ~1.5mbit/s bitrate.



Credit

2 Projects that led the way

- https://tinypilotkvm.com
- https://github.com/pikvm/pikvm

- Both use similar hardware (pi4) but stream Jpegs over http (over VPN)
- No code derived from either project - just inspiration
- Instead I leveraged |pipe|'s IoT Video stack

Big picture…


WebRTC
NAT/ Firewall
Public Internet
NAT/ Firewall
Raspberry Pi 4
Web Browser with WebRTC
|pipel - muxcam - Live
|pipe| WebRTC Agent
H264 rtp
Gstreamer
USB3
Video Capture
HDMI
Keys
Server
Mouse/kbd Emulation
hidg0
hidg1
USB2
Keyboard and mouse
USBC

Big picture…

WebRTC
NAT/ Firewall
Public Internet
NAT/ Firewall
Raspberry Pi 4
Web Browser with WebRTC
|pipe| WebRTC Agent
H264 rtp
Gstreamer
USB3
Video Capture
HDMI
Keys
Server
Mouse/kbd Emulation
hidg0 hidg1
USB2
Keyboard and mouse
USBC
"Air Gap"

Air Gap (video)

No, not really but close….

- HDMI capture card ($14)
  - Cheap, dumb, predictable, v4l2 compatible
  - To linux it is a 1080p @30 camera device
  - USB (although CSI interfaces possible)
  - Pi has hardware h264 encoder that supports 1080p @30
  - bitrate 10% of mjpeg in typical useage (~2mbit/s vs 20)

Big picture…

"Air Gap"

Air Gap (keyboard+mouse)

No, not really but close….

- Pi 4 supports gadget mode on USB C port (also on usb of pi zero W)
  - HID emulation
  - Keyboard and mouse
  - With config device appears as /dev/hidg{01}
  - Simple write of 4 or 8 bytes to device produces kbd emulation
- Server can’t tell this isn’t a keyboard + monitor + mouse

Big picture…


WebRTC
NAT/ Firewall
Public Internet
NAT/ Firewall
Raspberry Pi 4
Web Browser with WebRTC
|pipe| WebRTC Agent
H264 rtp
Gstreamer
USB3
Video Capture
HDMI
Keys
Server
Mouse/kbd Emulation
hidg0
hidg1
USB2
Keyboard and mouse
USBC

Transport Protocol -> WebRTC

Don’t roll your own cryptography or protocols!

- Available on all browsers
- No client install needed (so no new risks at client end)
- Well studied protocol
- E2E encrypted with self-signed x509s for auth
- Built for low latency/high quality video
- Used a lot for screenshares - so geared up for this
- WebRTC traffic is expected on networks

WebRTC security properties

Default secure.

- No open ports until message exchange
- Open port is random
- Works behind NAT - so no IP to scan
- Opened port is protected by otp
- Selected port is verified with DTLS handshake
- DTLS extension uses key material to derive media session key
- Provided SCTP channels over DTLS for non-media data

Big picture…


WebRTC
NAT/ Firewall
Public Internet
NAT/ Firewall
Raspberry Pi 4
Web Browser with WebRTC
|pipe| WebRTC Agent
H264 rtp
Gstreamer
USB3
Video Capture
HDMI
Keys
Server
Mouse/kbd Emulation
hidg0
hidg1
USB2
Keyboard and mouse
USBC

|pipe| WebRTC agent

Cleanroom implementation for small linux devices

- In Java because:
  - Strong typing prevents many vulns
  - Buffer overflow protection
  - Stack smashing protection
  - Mature ecosystem (tools etc)
  - Performant on small machines

Defence in depth

Extra steps taken in |pipe| WebRTC agent

- No reflection - config files can’t control object creation
- Input parser does string compares not regexps
- Only exchanges packets with known peers
- Only opens media sessions with permitted known peers
- Permitted peers must have public key in local keystore
- Acts as a proxy for local service - via sockets not libs
- GitHub dependency alerts for upstream CVEs

Auth

Self signed Certs with proximity verification

- Auth is decentralized
- X509s created, stored and checked locally
- Exchanged using DTLS handshake
- Validated by nonce in a QR code (proof of proximity)
- QR shown on hdmi of Pi.
- QR scanned on laptop or phone/iPad
- Access can then be lent to other devices

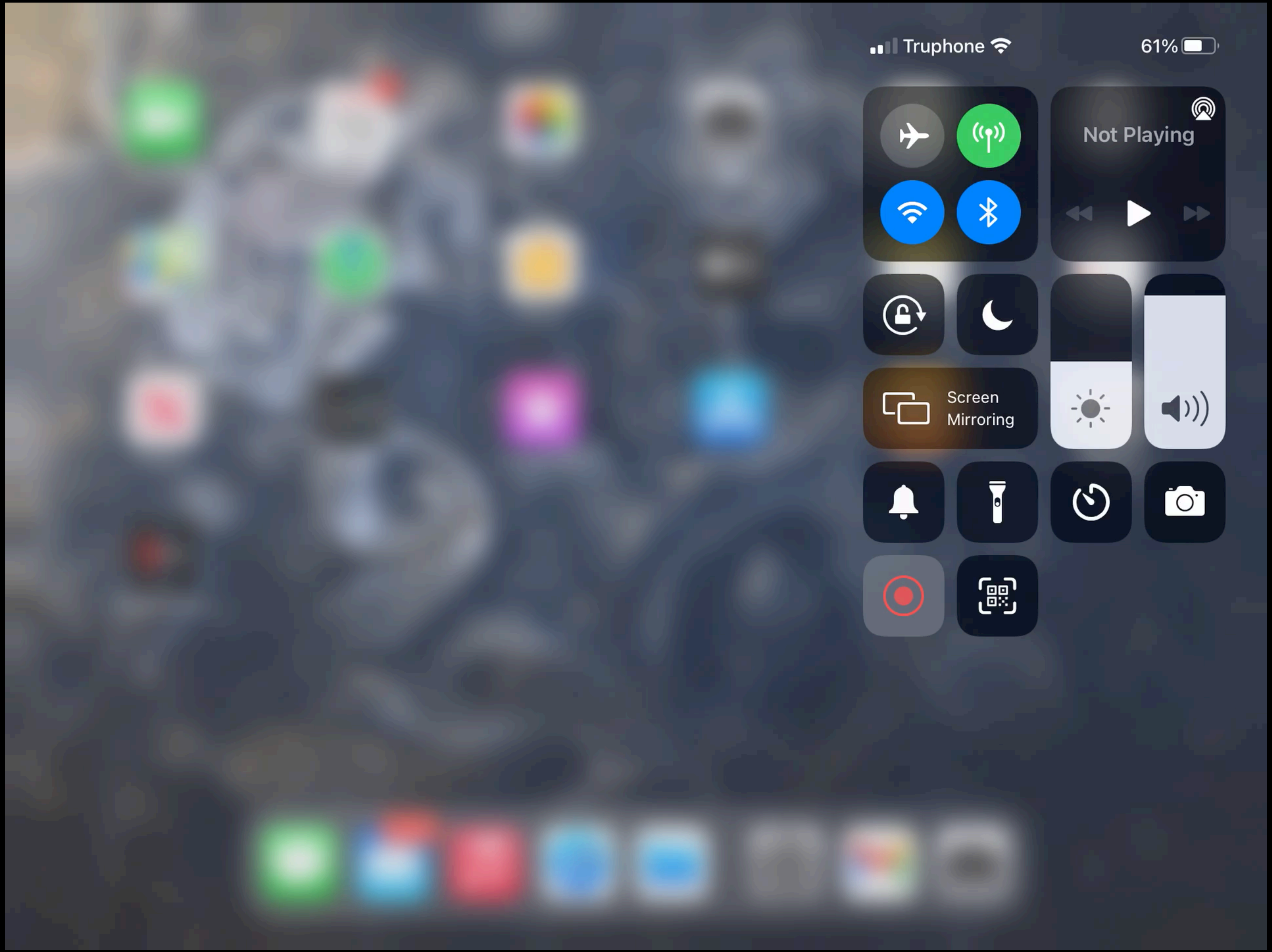
Proximity as proof of ownership

Things as tokens.

- Often we use centralized services to generate permission tokens Kerberos etc.
- In IoT we can use a Thing as a token.
- The owner is the first person to plug it in
- How does the device know it is you?
- Offers localized one time cryptographic handshake
- E.g scan a QR to prove you have Line of Sight

Sounds complex

But it looks like this….



But the Signalling ….

Ah, yes, that.

- We do need a ‘cloud’ service for connection establishment
- Not trusted with private data or keys
- RDV server (web server on known public IP)
    - Both ends connect to it over websocket
    - Exchange setup messages
    - Using hash of public key as an address (immune to iteration attacks)
    - Devices ignore setup messages that aren’t from permitted peers.
- Public key is tested as part of DTLS handshake so RDV can’t be MITM

How do they get throug NAT?

ICE is a WebRTC feature

- Uses a mix of tricks (simplified)
    - STUN allows each end to learn public IP
    - TURN service acts as a packet reflector if no direct path is available
    - Setup messages contain discovered public IPs + private IPs + IPv6
    - ICE tries all combinations of IPs + TURN until it finds a path that works
    - ICE secured by otp exchange

Metadata

Metadata collects in these places

- Webserver knows IPs + Times of usage
- STUN server knows IPs + Usage Time + duration
- TURN server knows user’s IP and Usage Time
- RDV server knows IPs, public keys of both ends
- KVM knows IPs, time,duration,public keys,keystrokes,video etc.

What we haven't done

Yet…

- Adversarial testing
- Fuzzing
- 3rd party code reviews

Known Flaws

Things we don’t like but can’t fix

- Raspi has No secure boot - a passerby could swap sd cards in seconds.
  - mitigation is probably a better case with the card inside
  - or hot glue :-)
- Gadget mode also supports usb ethernet emulation so a hacked Raspi could MiTM traffic.
  - This isn’t a bigger risk than a hacked pi using USB keyboard to change routing tables
- Perfect place to install a keylogger….
- Have to trust the website that loads the page

Think about security all the time

- Security is compatible with usability
- If you include it early enough in the process
- Keep it in mind all the way through
- Expose your design and developer teams to security thinking
- Expose your security teams to design and product thinking
- Compliance isn't enough.

Fin

Thanks for listening

- Contact tim@pi.pe or @steely_glint (twitter)
- Most of this is licensable for security cameras etc
- Questions?