OWASP IoT Top 10

A gentle introduction and an exploration of root causes

Hi!

Nick Johnston (@nickinfosec)

Currently: Coordinator, Sheridan College’s Bachelor of Cybersecurity

Previously: Digital forensics, incident response, pentester, developer

Recently: Maker stuff, learning electronics

Overview

- Motivations
- IoT Top 10 Intro
- ~~Case Study~~ ~~Dirty Hack~~ Experiment
- Findings
- Solutions?
- Q&A

Won’t be talking about

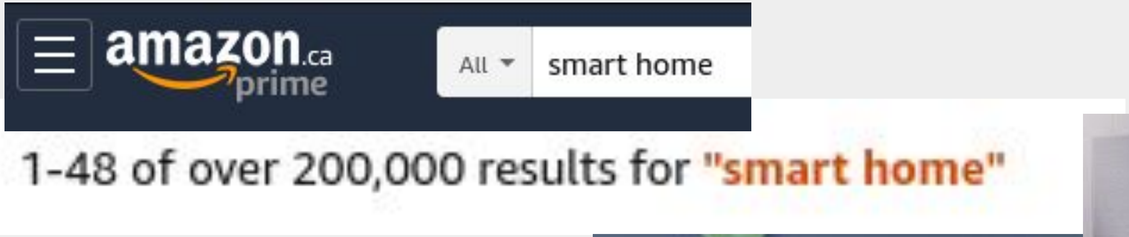
Manufacturing supply chain attacks (*that* Bloomberg article)

Non-consumer IoT:

- ICS/SCADA
- Medical
- Military

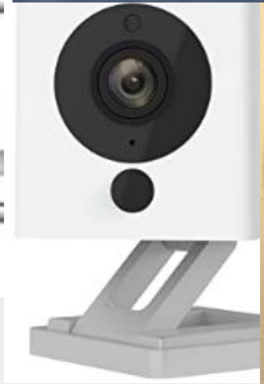
Impact of vulnerabilities

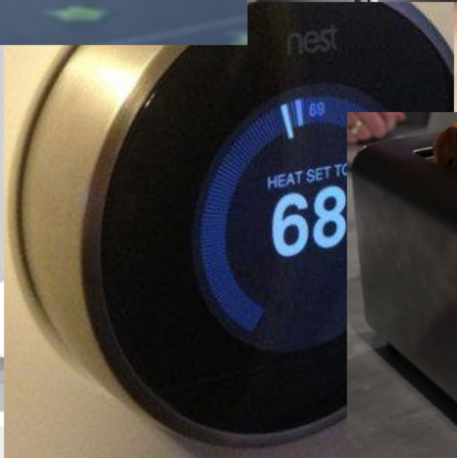
CONNECT ALL THE THINGS!

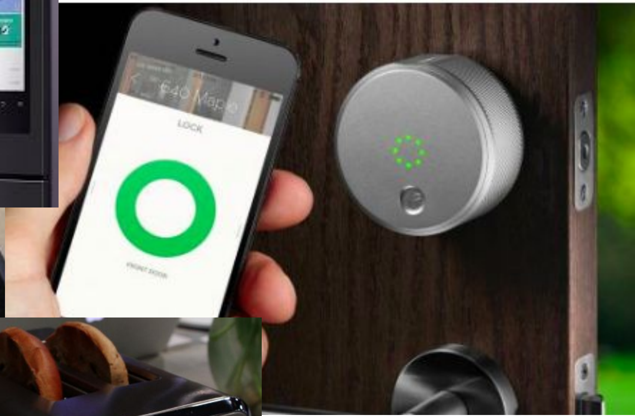

amazon.ca prime

All smart home

1-48 of over 200,000 results for "smart home"







The Cost of Convenience

threatpost

2 Million IoT Devices Vulnerable to Complete Takeover

Hack of High-End Hotel Smart Locks Shows IoT Security Fail

CVE Details

The ultimate security vulnerability datasource

Belkin » Wemo Home Automation Firmware

Code Execution
1

CONSUMERAFFAIRS

Security researchers from Pen Test Partners speaking at the DEF CON Hacking Conference announced their discovery of a security exploit that leaves Samsung model RF28HMELBSR smart refrigerators vulnerable to man-in-the-middle attacks (which allow hackers to alter, spy on, or control data while it's traveling between the sender and receiver).

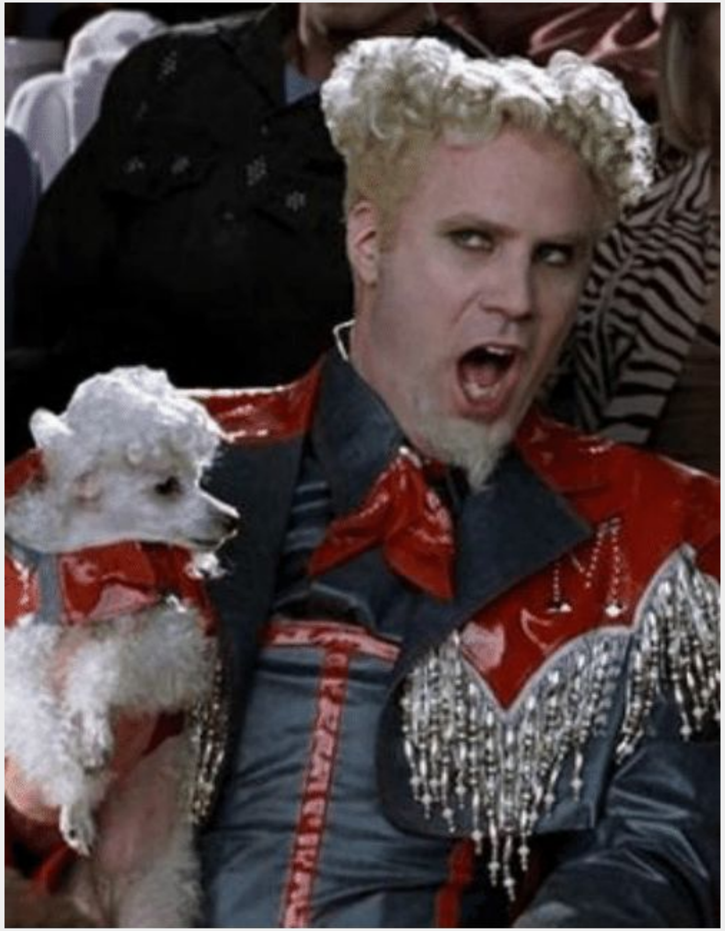
Motivations

IoT Security Is So Hot Right Now

- BlackHat 2017 - 8 Talks
- BlackHat 2018 - 14 Talks
- BlackHat 2019 - 8 Talks

OWASP IoT Top 10 - 2018

I like electronics and cybersecurity



Primary Motivation - SecTor 2019

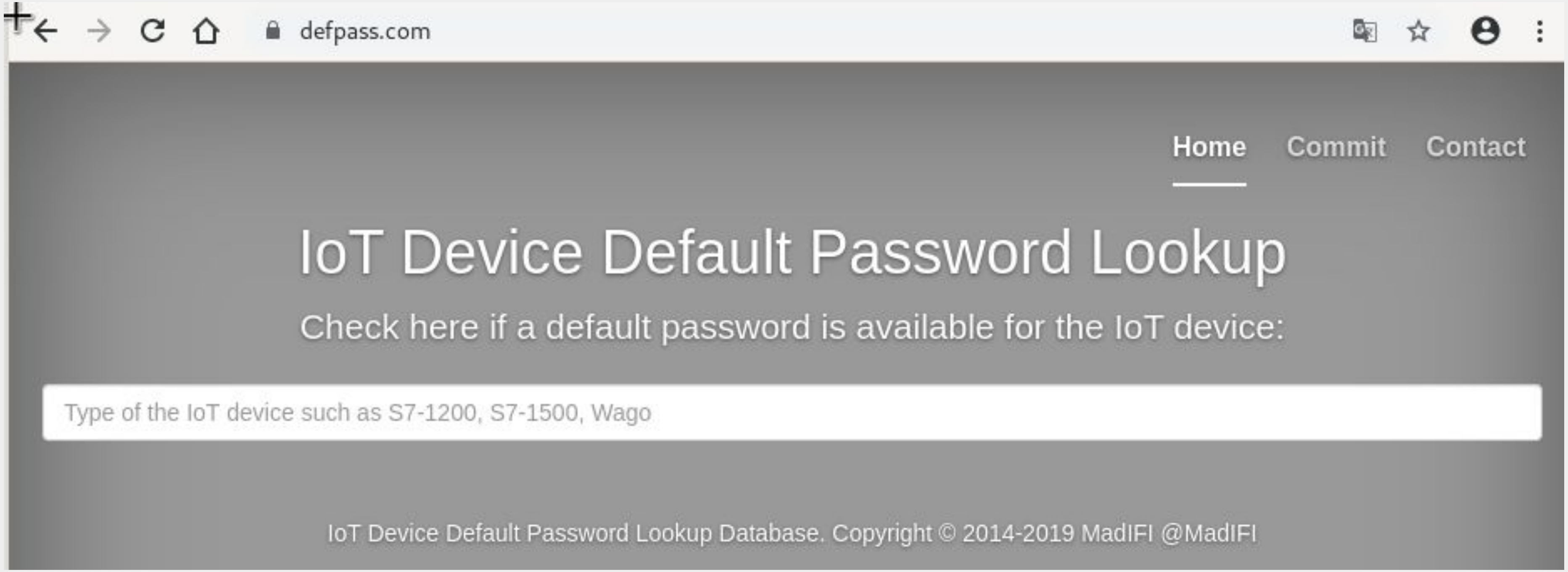
Lee Brotherston - “IoT Security: *An Insider's Perspective*”
https://sector.ca/sessions/iot-security-an-insiders-perspective/

- $things in $places (aka. *The Warehouse Problem*)
- Identity and Access Management (IAM)
- Low Friction Deployment
- Software Supply Chain
- Hardware protections are not feasible for consumer IoT
- Revenue challenges

OWASP IoT Top 10

https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project

1. Weak, Guessable, or Hardcoded Passwords


defpass.com

Home Commit Contact

IoT Device Default Password Lookup

Check here if a default password is available for the IoT device:

Type of the IoT device such as S7-1200, S7-1500, Wago

IoT Device Default Password Lookup Database. Copyright © 2014-2019 MadIFI @MadIFI

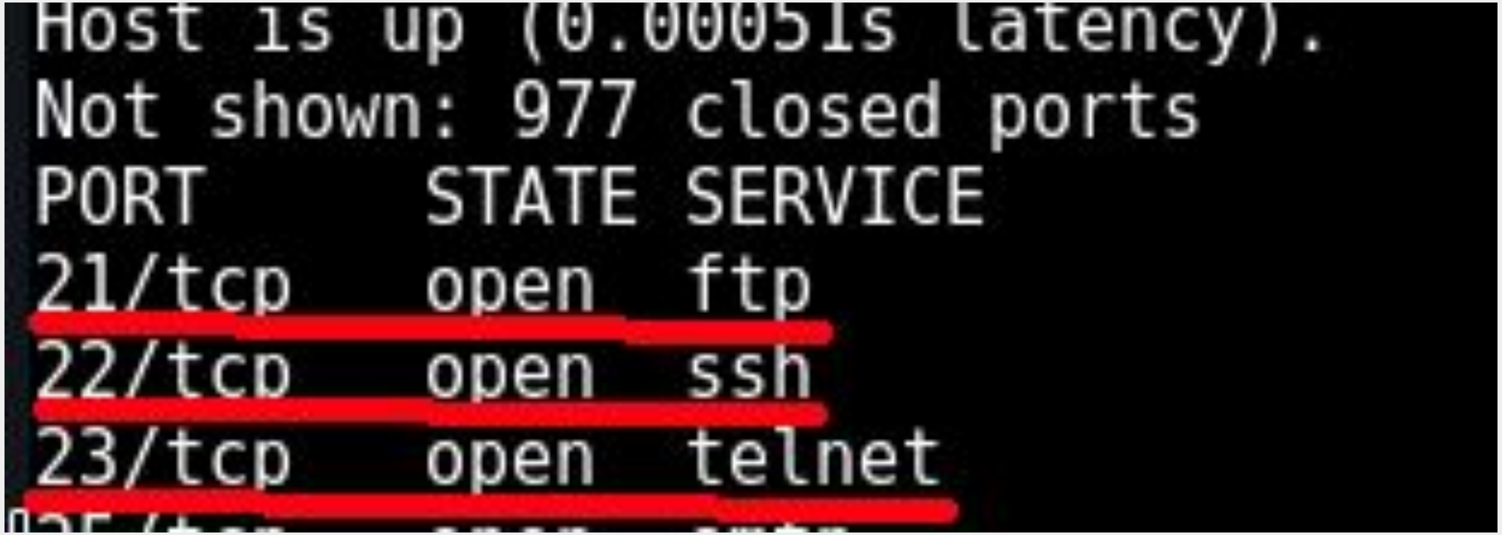
1. Weak, Guessable, or Hardcoded Passwords

Use of:

- Easily bruteforced
- Publicly available
- Unchangeable credentials

Including backdoors in firmware or client software that grants unauthorized access.

2. Insecure Network Services



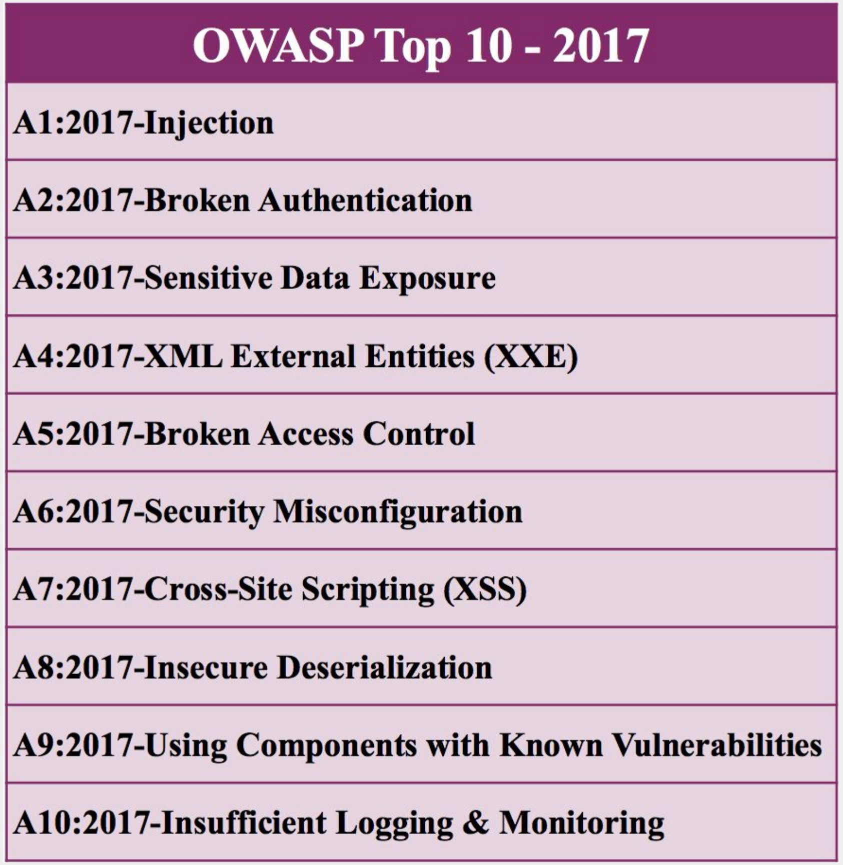
```
Host is up (0.00051s latency).
Not shown: 977 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
```

2. Insecure Network Services

Unneeded or insecure network services running on the device itself, especially:

- Those exposed to the Internet
- Any that compromise the confidentiality, integrity/authenticity, or availability of information
- Any service that allows unauthorized remote control

3. Insecure Ecosystem Interfaces



OWASP Top 10 - 2017
A1:2017-Injection
A2:2017-Broken Authentication
A3:2017-Sensitive Data Exposure
A4:2017-XML External Entities (XXE)
A5:2017-Broken Access Control
A6:2017-Security Misconfiguration
A7:2017-Cross-Site Scripting (XSS)
A8:2017-Insecure Deserialization
A9:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring

I swear they didn't pay me to put this in here...

3. Insecure Ecosystem Interfaces

Insecure interfaces in the ecosystem outside the device:

- Web
- Backend API
- Cloud
- Mobile

Common issues:

- Lack of authentication
- Lack of authorization
- Lacking or weak encryption
- Lack of input and output filtering

4. Lack of Secure Update Mechanism

Lack of ability to securely update the device.

- Lack of firmware validation on device
- Lack of secure delivery (un-encrypted in transit)
- Lack of anti-rollback mechanisms
- Lack of notifications of security changes due to updates

4. Lack of Secure Update Mechanism

2016 Carnegie Mellon University Study

On Board Diagnostics: Risks and Vulnerabilities of the Connected Vehicle

- Observations: insecure firmware updates and downloads
- Researchers were able to make arbitrary firmware modifications and maliciously update remote firmware.

https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=453871

5. Use of Insecure or Outdated Components

Use of deprecated or insecure software components/libraries that could allow the device to be compromised.

- Insecure customization of operating system platforms
- Third-party software libraries from a compromised supply chain
- Third-party hardware components from a compromised supply chain

5. Use of Insecure or Outdated Components

Meltdown

Spectre

Heartbleed

6. Insufficient Privacy Protection

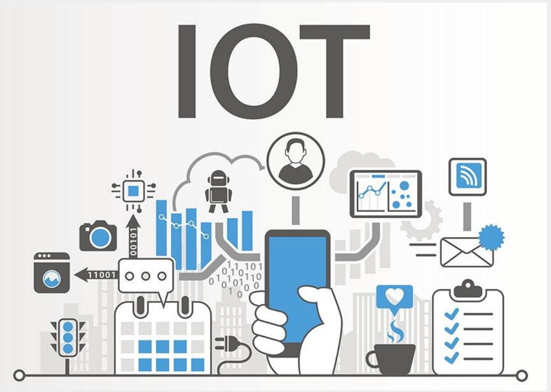
User’s personal information stored on the device or in the ecosystem that is used **insecurely**, **improperly**, or **without permission**.

6. Insufficient Privacy Protection

2017 Cornell University Study
A Smart Home is No Castle: Privacy Vulnerabilities of Encrypted IoT Traffic
“we examine four IoT smart home devices [...] and find that their network traffic rates can reveal potentially sensitive user interactions even when the traffic is encrypted”
https://arxiv.org/abs/1705.06805

7. Insecure Data Transfer and Storage

Lack of encryption or access control of sensitive data anywhere within the ecosystem, including at rest, in transit, or during processing.



+ Let's Encrypt =

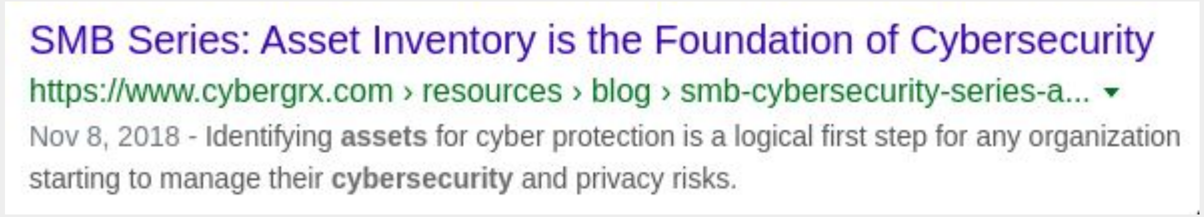
7. Insecure Data Transfer and Storage

“The Espressif **ESP8266** chipset makes three-dollar ‘Internet of Things’ development boards an economic reality. According to the popular automatic firmware-building site nodeMCU-builds, in the **last 60 days** there have been **13,341 custom firmware builds** for that platform. Of those, **only 19% have SSL support**, and 10% include the cryptography module.”

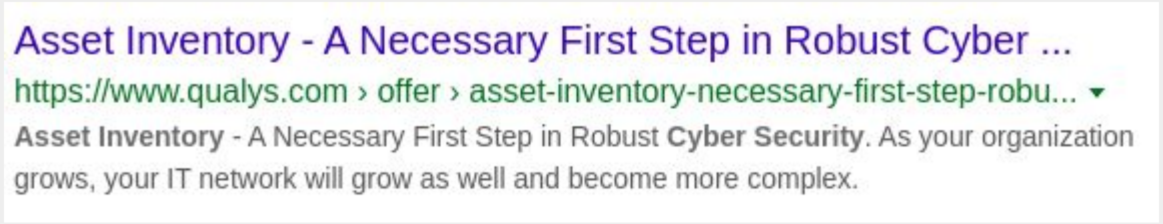
https://hackaday.com/2017/06/20/practical-iot-cryptography-on-the-espressif-esp8266/

8. Lack of Device Management

Lack of security support on devices deployed in production, including asset management, update management, secure decommissioning, systems monitoring, and response capabilities.



SMB Series: Asset Inventory is the Foundation of Cybersecurity
https://www.cybergrx.com › resources › blog › smb-cybersecurity-series-a...
Nov 8, 2018 - Identifying **assets** for cyber protection is a logical first step for any organization starting to manage their **cybersecurity** and privacy risks.



Asset Inventory - A Necessary First Step in Robust Cyber ...
https://www.qualys.com › offer › asset-inventory-necessary-first-step-robu...
Asset Inventory - A Necessary First Step in Robust **Cyber Security**. As your organization grows, your IT network will grow as well and become more complex.

8. Lack of Device Management

We haven’t solved this for non-IoT environments yet..

- 25% still rely on Excel spreadsheets to track assets
- 56% verify asset location only once a year, while 10-15% verify only every five years
- Staff spends 10+ hours weekly to resolve data accuracy issues
- Nearly 66% of IT managers have an incomplete record of their IT assets

https://www.scmagazine.com/home/opinion/executive-insight/tighter-control-over-it-asset-management-the-key-to-securing-your-enterprise/

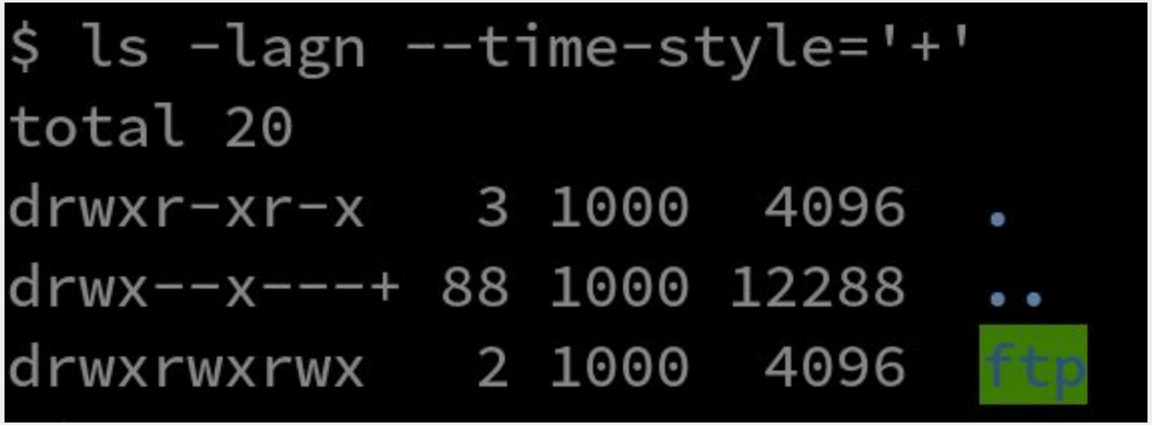
9. Insecure Default Settings

Devices or systems shipped with insecure default settings or lack the ability to make the system more secure by restricting operators from modifying configurations.

9. Insecure Default Settings

Bad filesystem permissions

Exposed services running as root



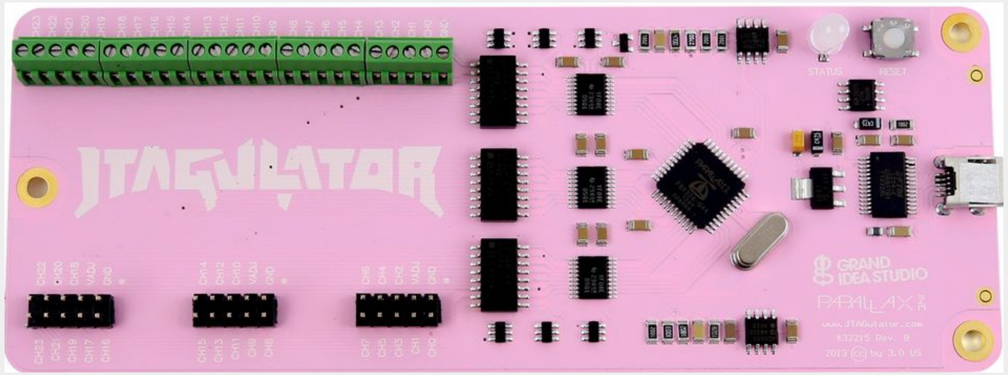
```
$ ls -lagn --time-style='+'
total 20
drwxr-xr-x   3 1000  4096  .
drwx--x---+ 88 1000 12288  ..
drwxrwxrwx   2 1000  4096  ftp
```

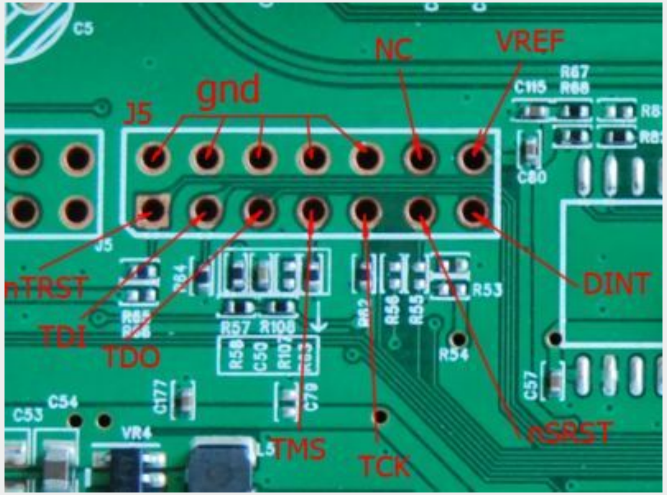
10. Lack of Physical Hardening

Lack of physical hardening measures, allowing potential attackers to gain sensitive information that can help in a future remote attack or take local control of the device.

10. Lack of Physical Hardening

Easily Available Debug Port Discovery





The Experiment

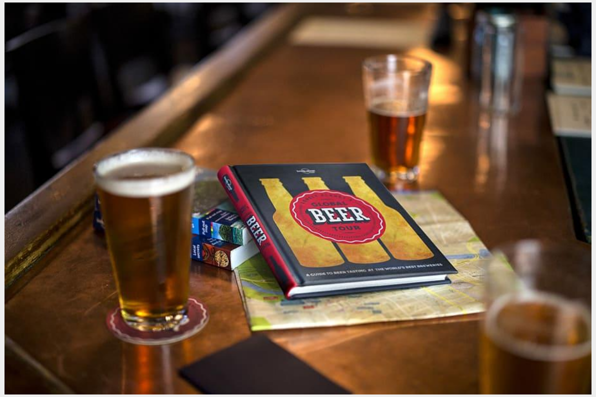
Wanted to identify potential root causes

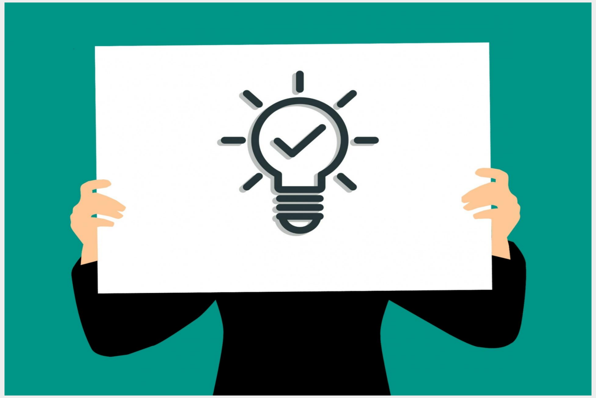
Wanted to simulate:

- Pressures of getting to market quickly
- Unfamiliarity with IoT product development process
- Unfamiliarity with secure development practices

A 24 hour IoT Hackathon

The Background (because we all love a narrative)





At the pub after work

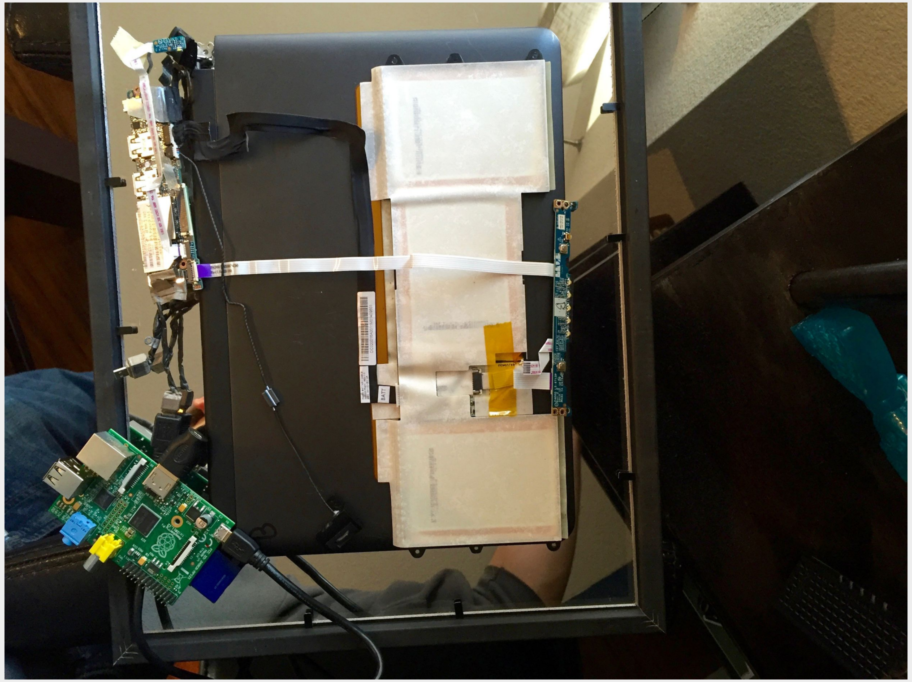
Get ~~website~~ IoT product drunk

Smart Mirrors!

What is a Smart Mirror?

A monitor and a Raspberry Pi taped to the back of a one-way mirror.

The Pi updates the display with some predetermined info like date/time, weather, train schedule, etc.

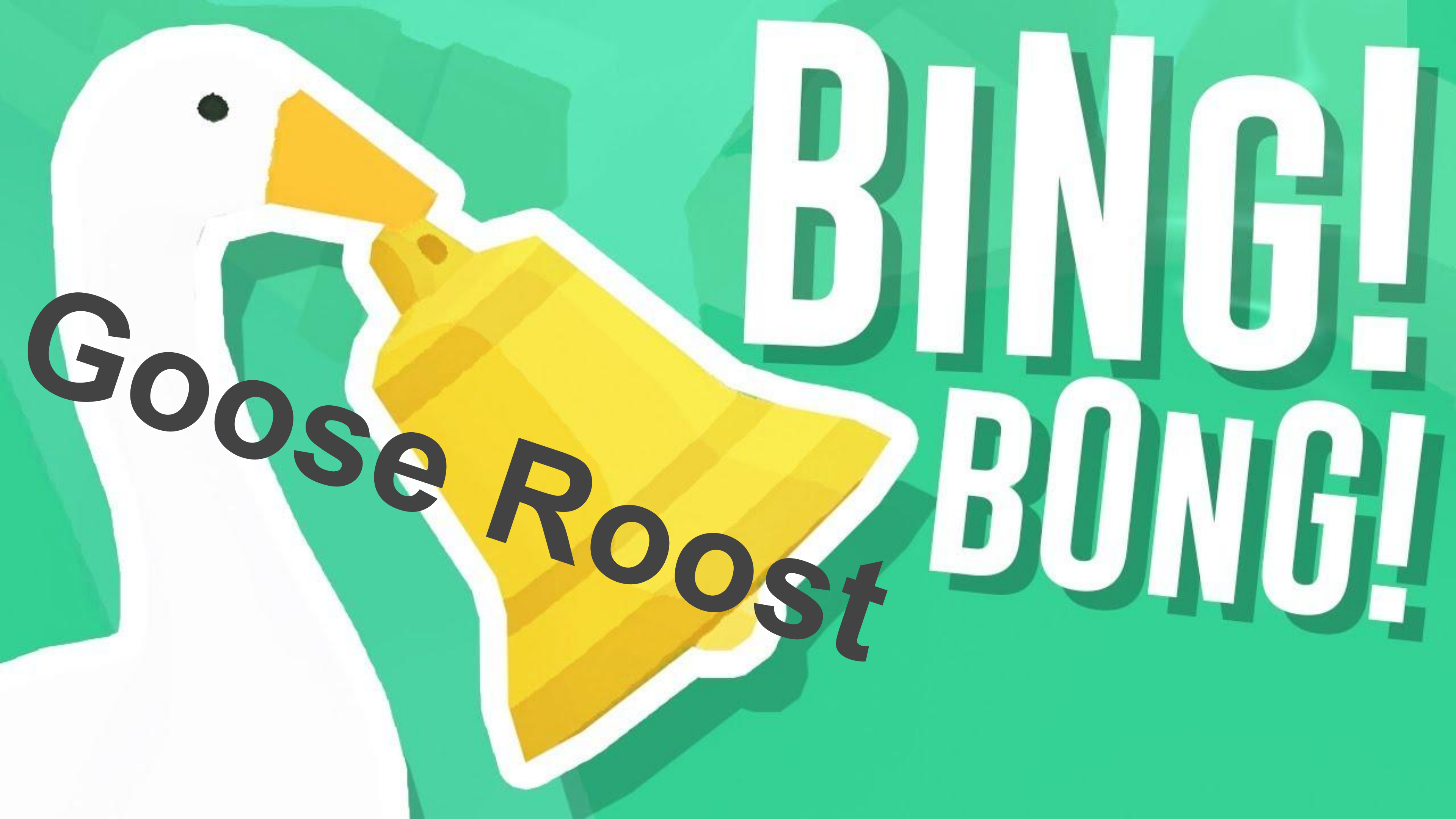


Other people are making smart mirrors!

I NEED to be FIRST for that sweet VC $$$.

My friend works for PrimeHuFlix+ and they got me a spot TOMORROW on ~~Dragons' Den~~ ~~Shark Tank~~ ...

Goose Roost



I get excited and start thinking about marketing...

I pick a hip name: Brainmirror

I “register a domain”

```
echo “localhost brainmirror.com” >> /etc/hosts
```

I work memes into your logo

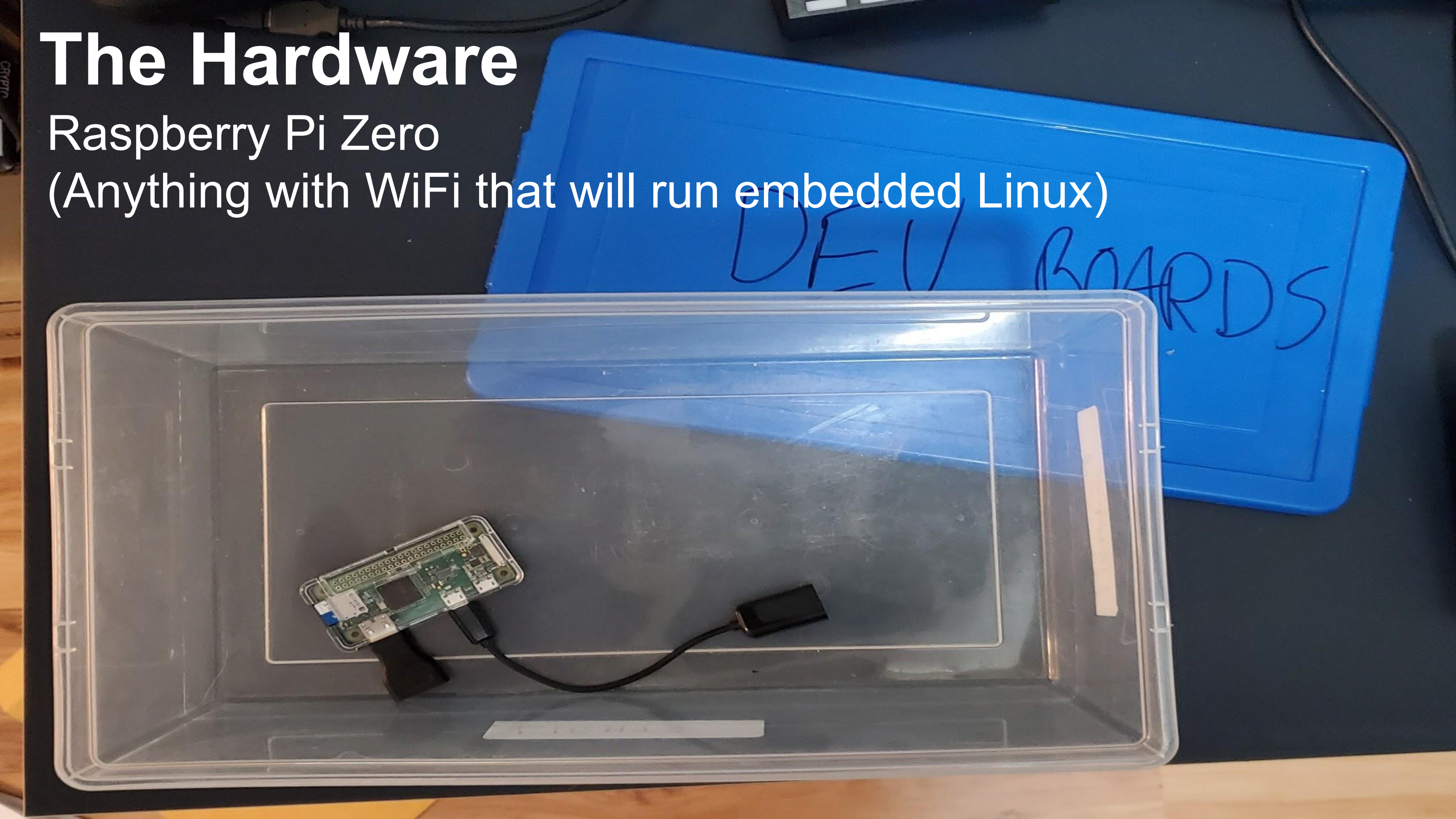
Oh wait...I have to make it first

Design Requirements

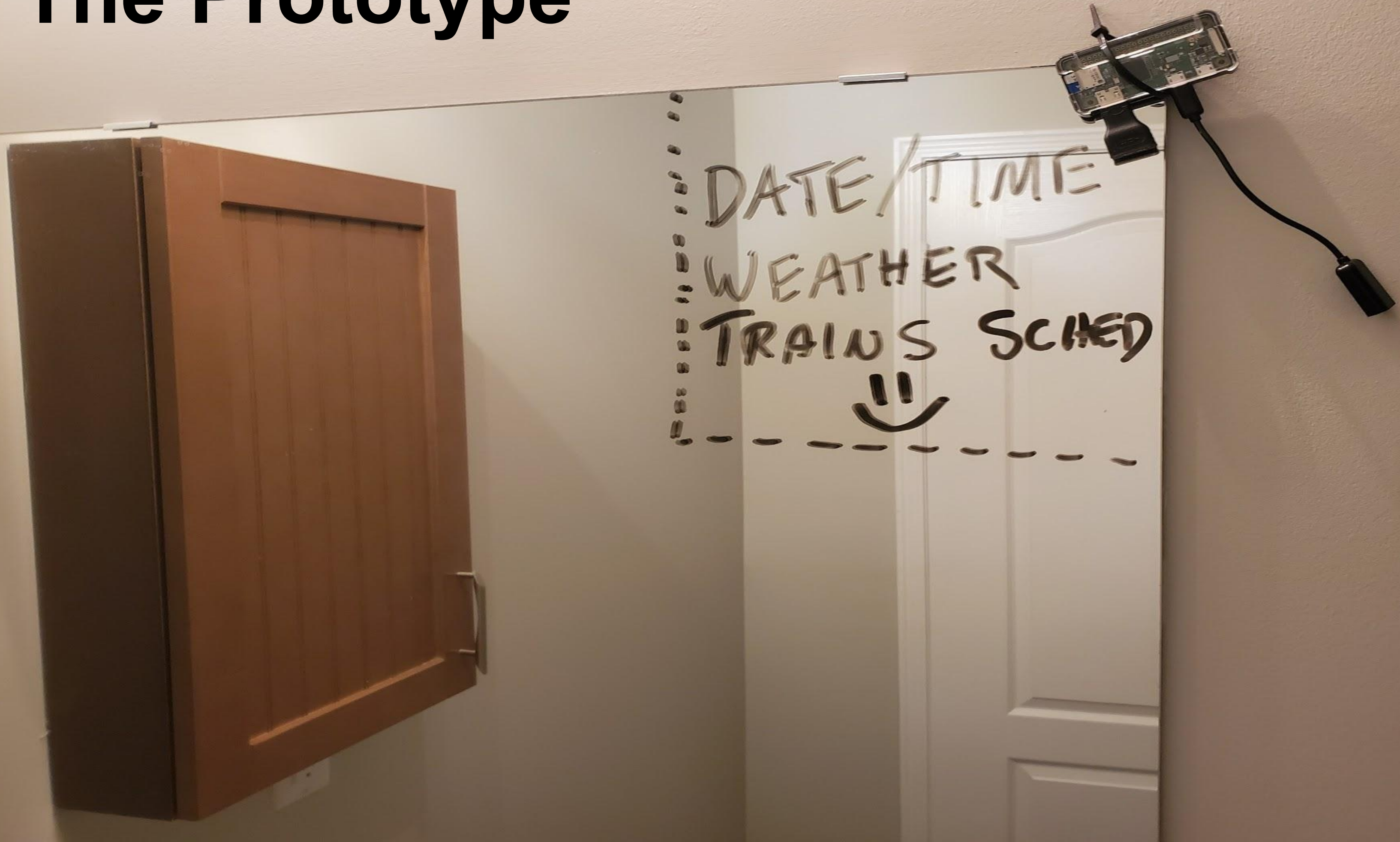
- Cheap
- No subscription
- Low friction deployment
- Ease of use
- (also it works..hopefully)

The Hardware

Raspberry Pi Zero
(Anything with WiFi that will run embedded Linux)



The Prototype



NETFLIX
Nailed It!
IoT Edition

General Solution Structure

1. Pi starts as a wireless access point
2. Connect to AP and enter local WiFi credentials
3. Device redirects to local setup/registration page
4. Registration page sent to server
5. Device reboots and starts fullscreen mirror application
6. Device queries remote server for data and updates

Technology Stack

~~docker~~ ~~React~~

~~aws~~

~~PostgreSQL~~ ~~Linux EMBEDDED~~

{JSON} redis

Flask

Raspbian

DigitalOcean

Raspbian Setup (Development Setup)

Download Raspbian (https://www.raspberrypi.org/downloads/raspbian/)

Copy the Raspbian image onto an SD card (replace sdX with yours)

```
    dd bs=4M if=your_raspbian_image.img of=/dev/sdX conv=fsync
```

Boot the Pi and run through the standard Raspbian installer

When the Pi reboots after installation, open a terminal

```
    sudo apt install python3 pip3 flask dnsmasq hostapd
```

Shutdown the Pi and image the SD card

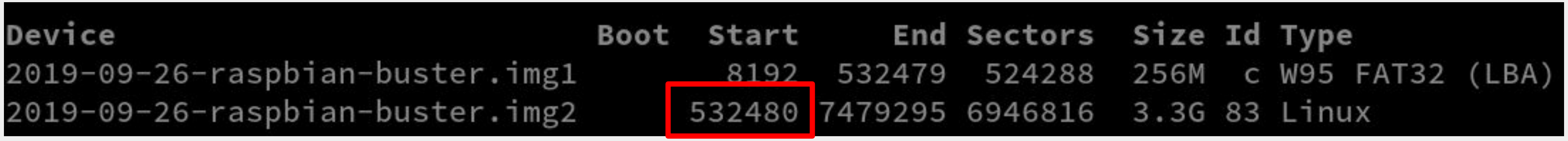
```
    dd bs=4M if=/dev/sdX of=dev_image.img
```

Raspbian Setup (Development Setup)

Now you can mount the image and edit any files, install the base software, etc.

Mounting the development image:

```
    sudo fdisk -l dev_image.img
```



```
Device                          Boot   Start     End Sectors  Size Id Type
2019-09-26-raspbian-buster.img1         8192  532479  524288  256M  c W95 FAT32 (LBA)
2019-09-26-raspbian-buster.img2       532480 7479295 6946816  3.3G 83 Linux
```

```
    532480 * 512 = 272629760

    sudo mkdir /mnt/pi
sudo mount -v -o offset=272629760 -t ext4 ./dev_image.img /mnt/pi
```

Copy application to /mnt/pi/app/brainmirror and edit configs (see later slides).
Now you can DD your image onto 100s of SD cards for manufacturing and deployment!

Raspbian Setup (No login boot)

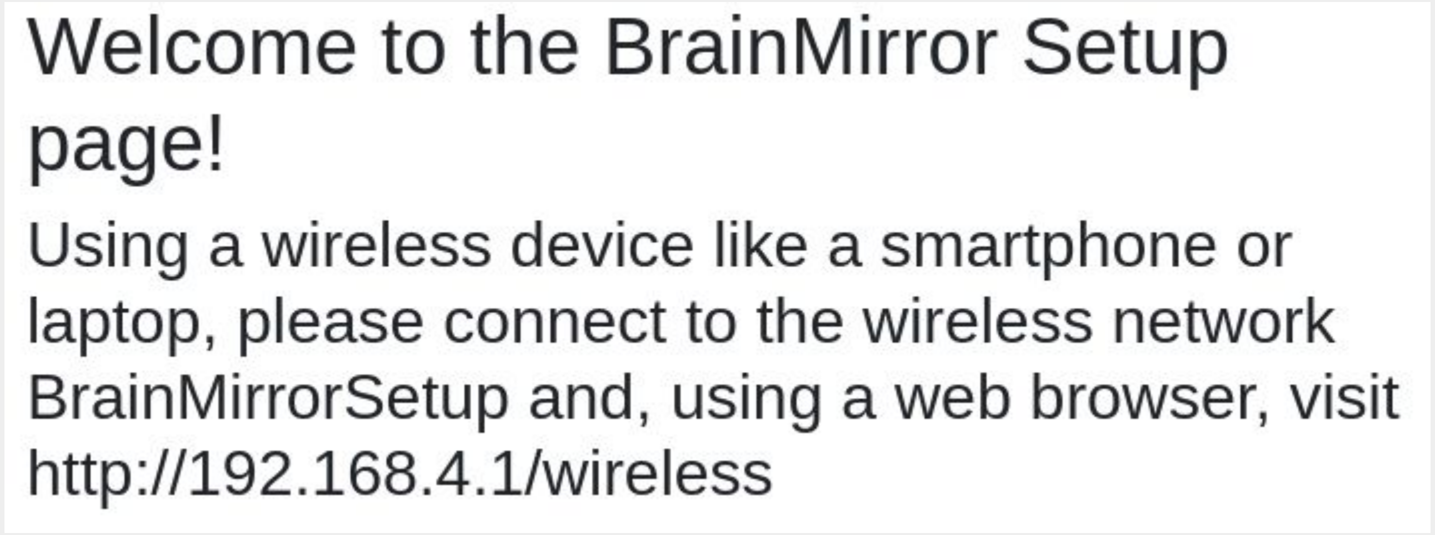
(The default is to boot to the desktop without a password prompt but maybe you want to boot to console and start X later? If so..)

```
$ vim /etc/inittab
#1:2345:respawn:/sbin/getty --noclear 38400 tty1
1:2345:respawn:/bin/login -f pi tty1 /dev/tty1 2>&1
:wq
$ sudo shutdown -r now
```

Raspbian Setup (Startup)

```
$ sudo vi /etc/rc.local
export FLASK_APP=wifi
flask run
if wificreds.txt exists
    sudo systemctl disable hostapd
    sudo systemctl stop hostapd
    chromium --app=file:///app/brainmirror/mirror.html \
        --start-fullscreen --kiosk
else
    # We’re running a wireless AP (see next few slides)
    chromium --app=file:///app/brainmirror/setup.html \
        --start-fullscreen --kiosk
```

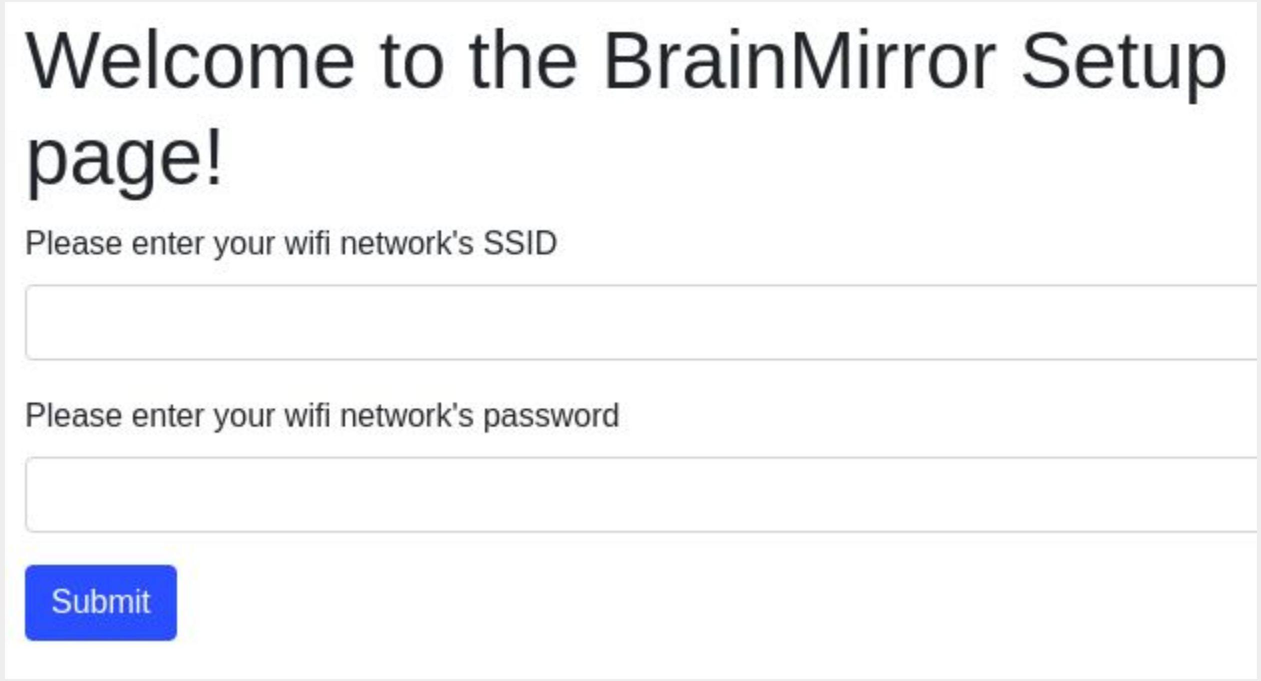
Setup.html (this will be displayed on the mirror)



Welcome to the BrainMirror Setup page!

Using a wireless device like a smartphone or laptop, please connect to the wireless network BrainMirrorSetup and, using a web browser, visit http://192.168.4.1/wireless

Wireless.html (this will be displayed on user’s phone)

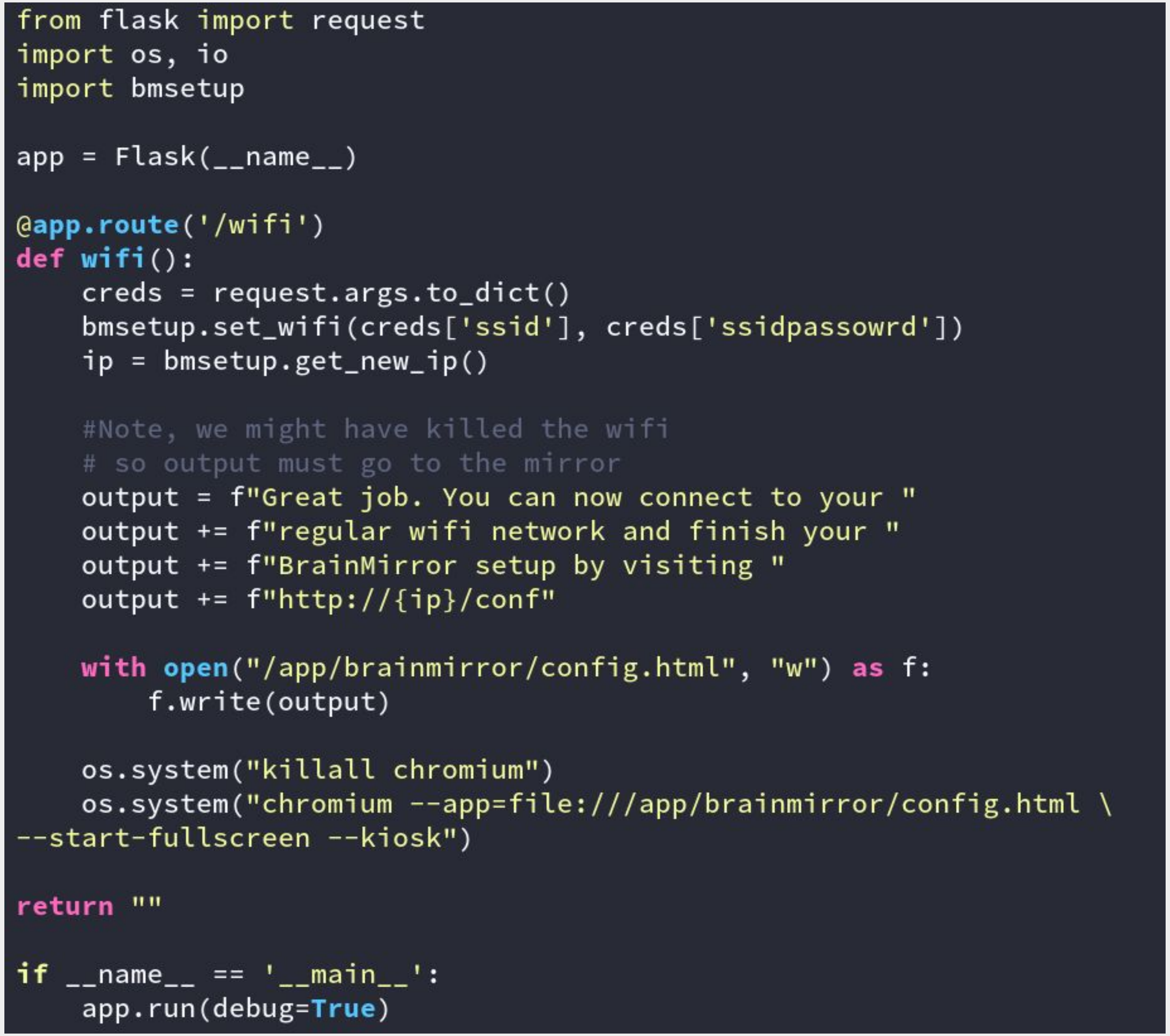

Welcome to the BrainMirror Setup page!

Please enter your wifi network's SSID

Please enter your wifi network's password

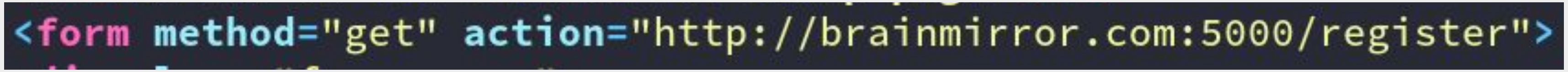
```
<form method="get" action="http://127.0.0.1:5000/wifi">
```

Response

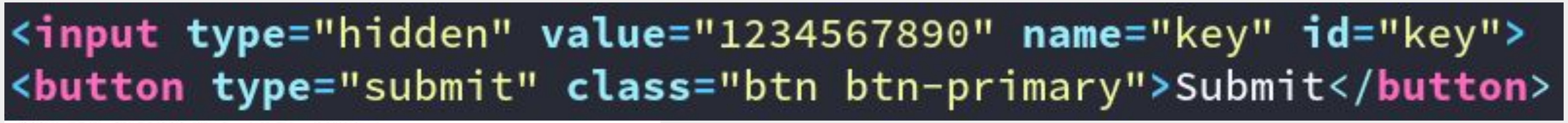


```python
from flask import request
import os, io
import bmsetup

app = Flask(__name__)

@app.route('/wifi')
def wifi():
    creds = request.args.to_dict()
    bmsetup.set_wifi(creds['ssid'], creds['ssidpassowrd'])
    ip = bmsetup.get_new_ip()

    #Note, we might have killed the wifi
    # so output must go to the mirror
    output = f"Great job. You can now connect to your "
    output += f"regular wifi network and finish your "
    output += f"BrainMirror setup by visiting "
    output += f"http://{ip}/conf"

    with open("/app/brainmirror/config.html", "w") as f:
        f.write(output)

    os.system("killall chromium")
    os.system("chromium --app=file:///app/brainmirror/config.html \
--start-fullscreen --kiosk")

return ""

if __name__ == '__main__':
    app.run(debug=True)
```

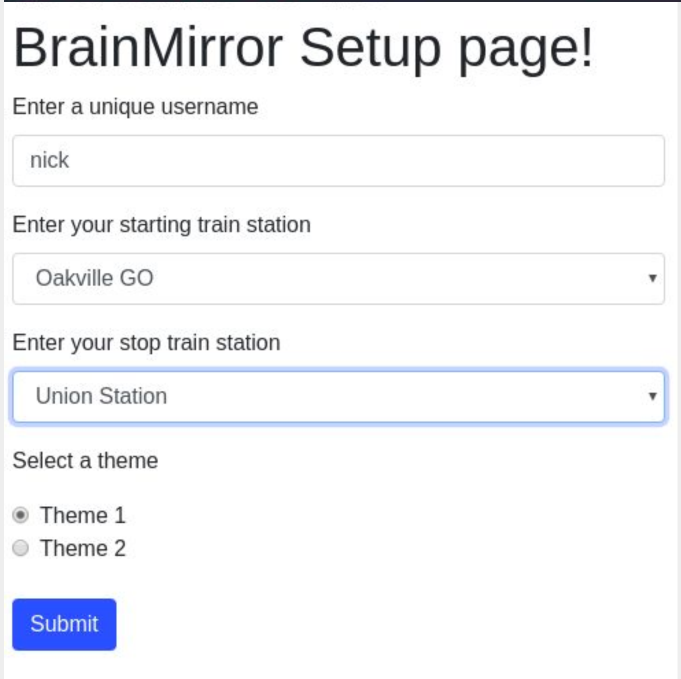
Client Setup - Registration Page



```
<form method="get" action="http://brainmirror.com:5000/register">
```



```
<input type="hidden" value="1234567890" name="key" id="key">
<button type="submit" class="btn btn-primary">Submit</button>
```



BrainMirror Setup page!

Enter a unique username

nick

Enter your starting train station

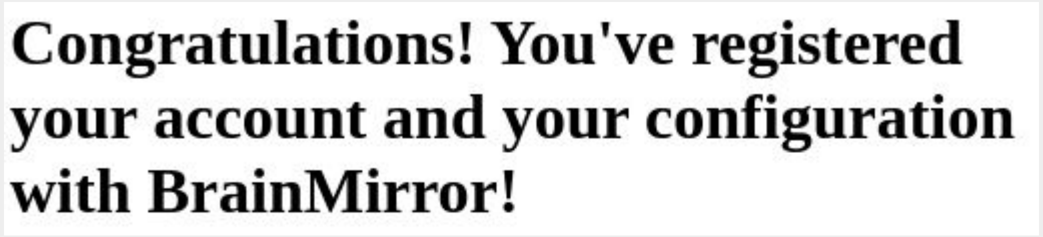
Oakville GO

Enter your stop train station

Union Station

Select a theme

- [✓ Theme 1]
- Theme 2



Congratulations! You've registered your account and your configuration with BrainMirror!

Raspbian Setup (Standalone AP)

```
$ sudo systemctl stop dnsmasq
$ sudo systemctl stop hostapd
$ sudo vi /etc/dhcpcd.conf
interface wlan0
    static ip_address=192.168.4.1/24
    nohook wpa_supplicant
:wq
$ sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.orig
$ sudo vi /etc/dnsmasq.conf
interface=wlan0
dhcp-range=192.168.4.2,192.168.4.20,255.255.255.0,24h
:wq
```

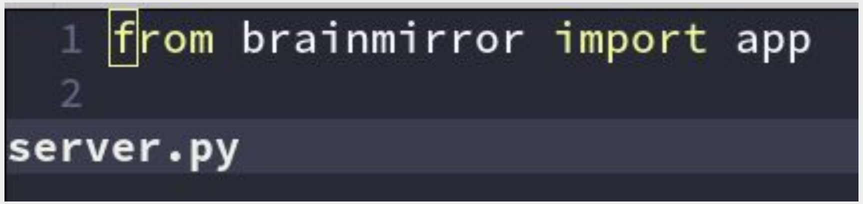
Raspbian Setup (Standalone AP - cont)

```
$ sudo vi /etc/hostapd/hostapd.conf
interface=wlan0
driver=nl80211
ssid=BrainMirrorSetup
channel=1
:wq
$ sudo vi /etc/default/hostapd
DAEMON_CONF="/etc/hostapd/hostapd.conf"
:wq
$ sudo systemctl unmask hostapd
$ sudo systemctl enable hostapd
```

Server Build (basically)

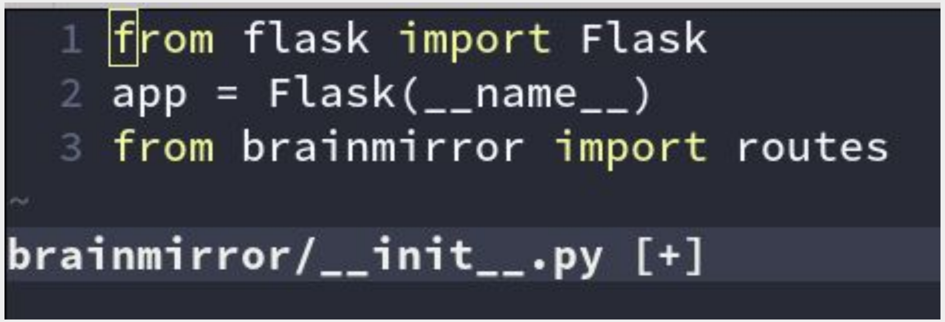
```
$ ssh nick@brainmirror.com

$ sudo apt install python3 pip3 redis git

$ git clone brainmirror; cd brainmirror

$ pip3 install -r requirements.txt

$ sudo cp brainmirror.service /etc/systemd/system/

$ sudo systemctl daemon-reload

$ sudo systemctl start brainmirror
```

Server-Side Code



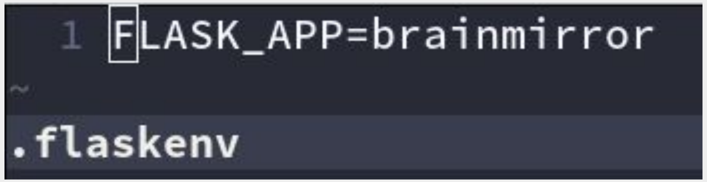
```python
from brainmirror import app

```

```
server.py
```



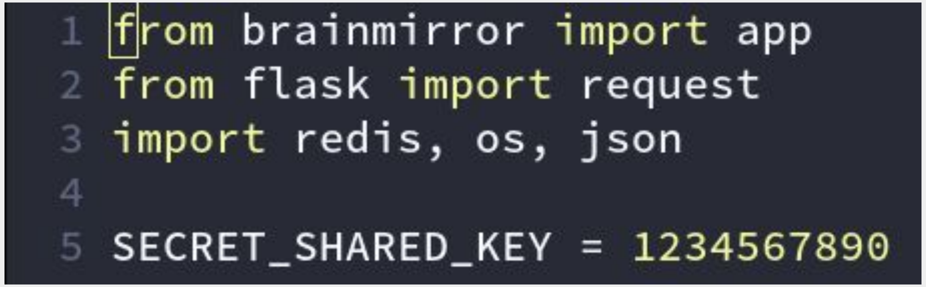
```python
from flask import Flask
app = Flask(__name__)
from brainmirror import routes
~
```

```
brainmirror/__init__.py [+]
```

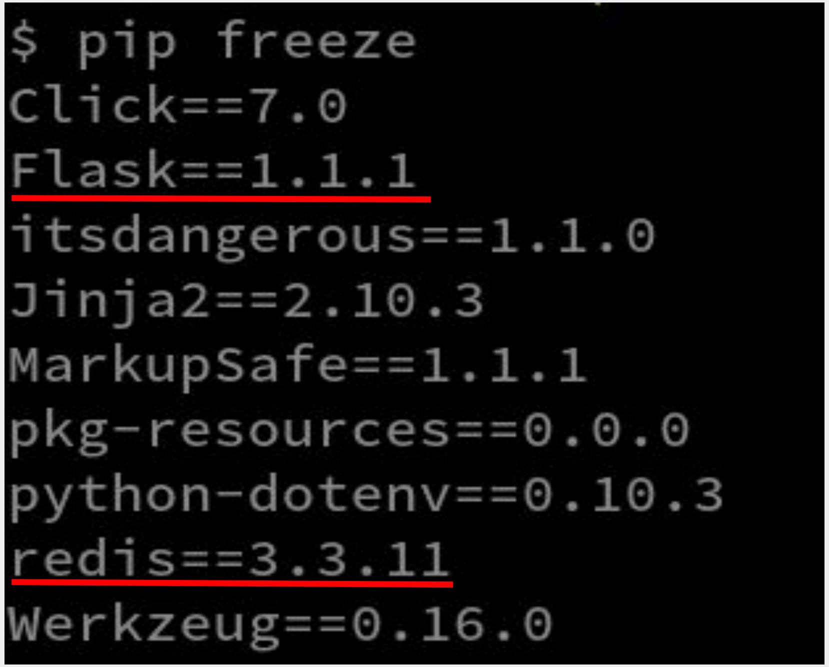


```
FLASK_APP=brainmirror
~
```

```
.flaskenv
```

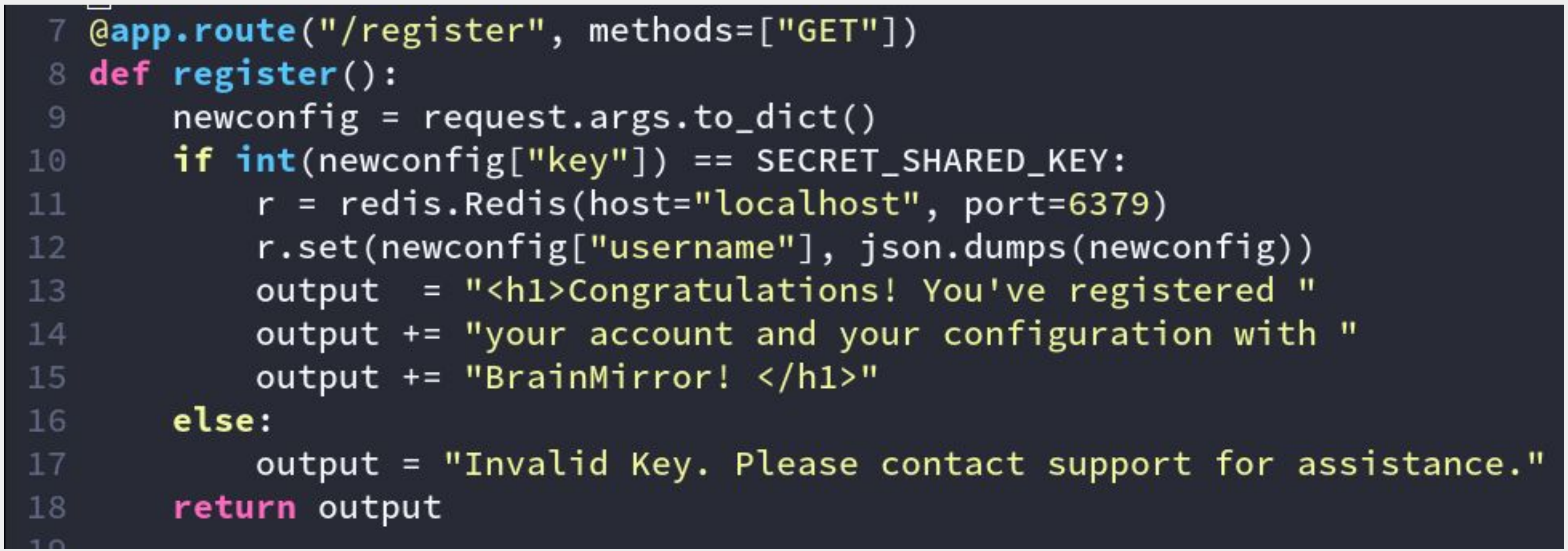


```python
from brainmirror import app
from flask import request
import redis, os, json

SECRET_SHARED_KEY = 1234567890
```



```
$ pip freeze
Click==7.0
Flask==1.1.1
itsdangerous==1.1.0
Jinja2==2.10.3
MarkupSafe==1.1.1
pkg-resources==0.0.0
python-dotenv==0.10.3
redis==3.3.11
Werkzeug==0.16.0
```

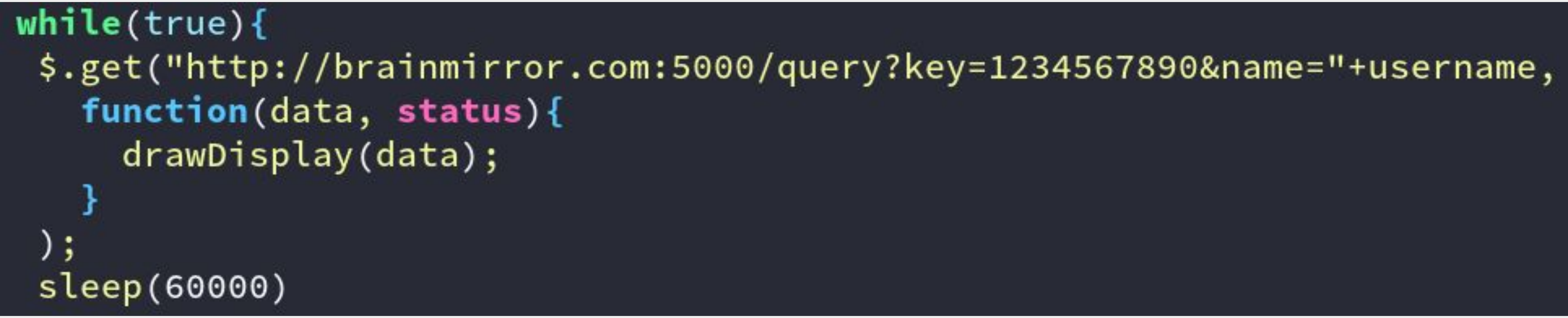
Server-Side Code - Device Registration



```python
@app.route("/register", methods=["GET"])
def register():
    newconfig = request.args.to_dict()
    if int(newconfig["key"]) == SECRET_SHARED_KEY:
        r = redis.Redis(host="localhost", port=6379)
        r.set(newconfig["username"], json.dumps(newconfig))
        output  = "<h1>Congratulations! You've registered "
        output += "your account and your configuration with "
        output += "BrainMirror! </h1>"
    else:
        output = "Invalid Key. Please contact support for assistance."
    return output
```

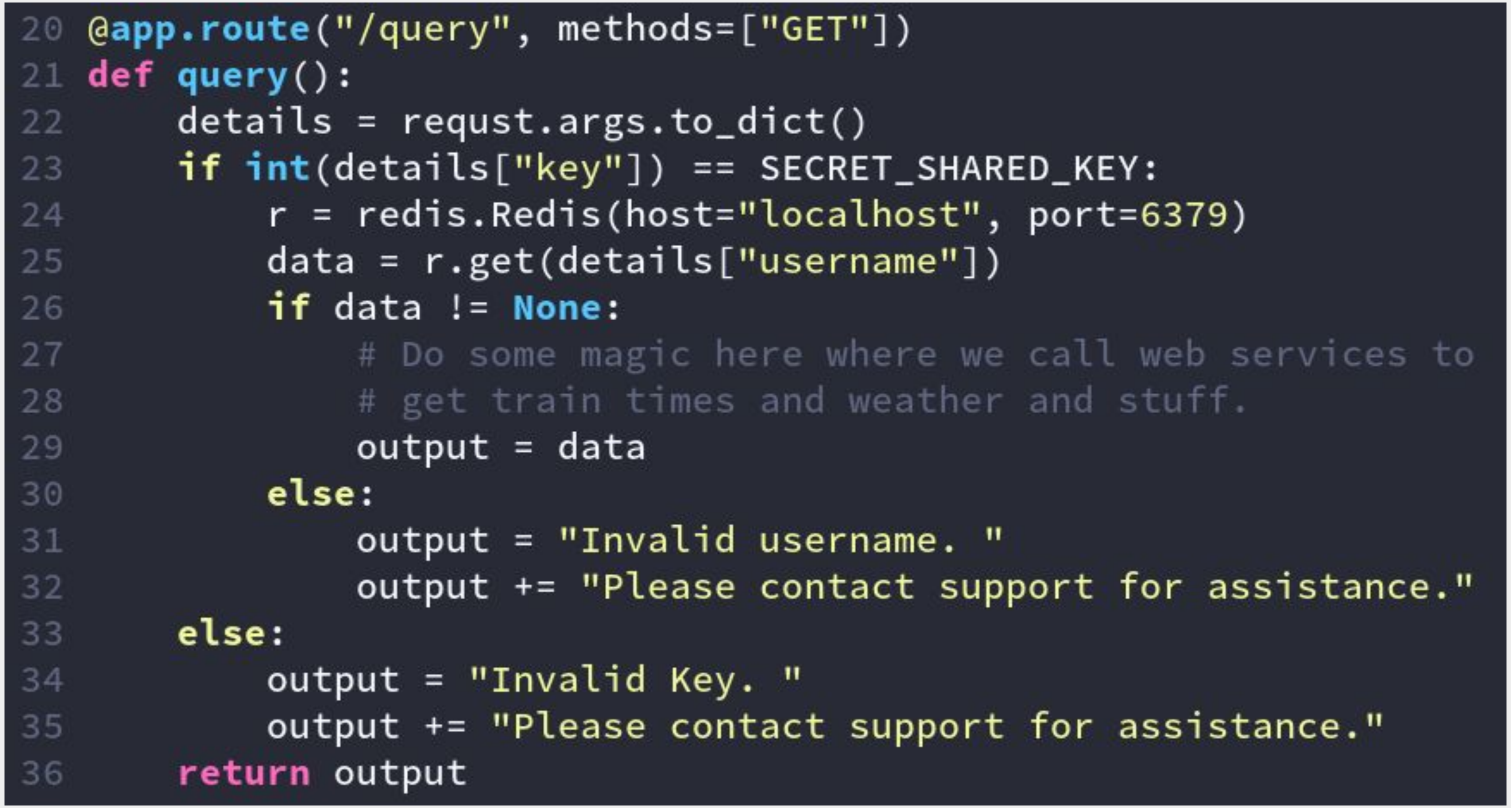
Mirror Code

`mirror.html` (the important bit)



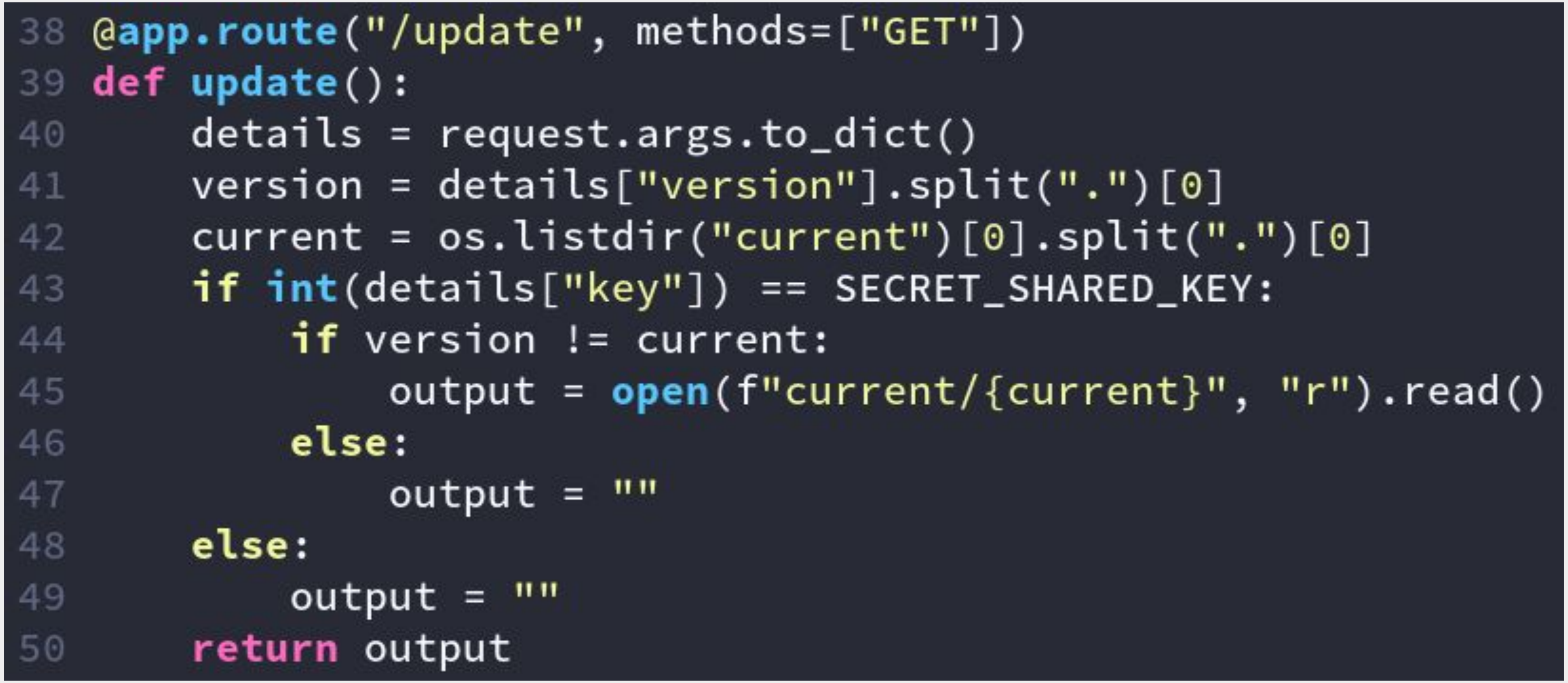
```
while(true){
 $.get("http://brainmirror.com:5000/query?key=1234567890&name="+username,
   function(data, status){
     drawDisplay(data);
   }
 );
 sleep(60000)
```

Server-Side Code - Getting Mirror Data



```python
@app.route("/query", methods=["GET"])
def query():
    details = requst.args.to_dict()
    if int(details["key"]) == SECRET_SHARED_KEY:
        r = redis.Redis(host="localhost", port=6379)
        data = r.get(details["username"])
        if data != None:
            # Do some magic here where we call web services to
            # get train times and weather and stuff.
            output = data
        else:
            output = "Invalid username. "
            output += "Please contact support for assistance."
    else:
        output = "Invalid Key. "
        output += "Please contact support for assistance."
    return output
```

Server-Side Code - Software Updates

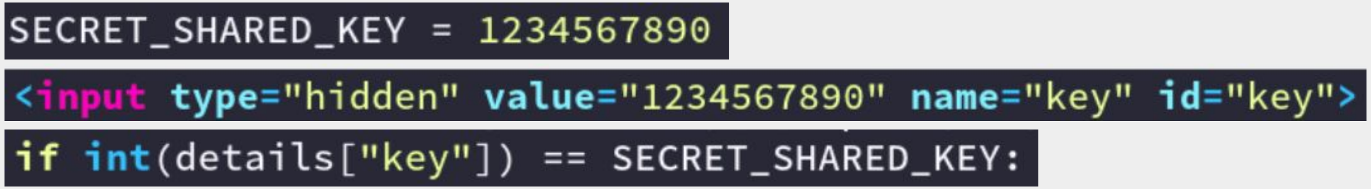


```python
@app.route("/update", methods=["GET"])
def update():
    details = request.args.to_dict()
    version = details["version"].split(".")[0]
    current = os.listdir("current")[0].split(".")[0]
    if int(details["key"]) == SECRET_SHARED_KEY:
        if version != current:
            output = open(f"current/{current}", "r").read()
        else:
            output = ""
    else:
        output = ""
    return output
```

I think I’ve made my point. We’ll just end this before it gets worse.

What went wrong?

1. **Weak, Guessable, or Hardcoded Passwords**



```
SECRET_SHARED_KEY = 1234567890
<input type="hidden" value="1234567890" name="key" id="key">
if int(details["key"]) == SECRET_SHARED_KEY:
```

Also we never changed the default Raspberry Pi user in Raspbian.

Why?

No idea how to do fancy “first time untrusted connection” protocols. It was easy to just make a shared key and it helps with “*The Warehouse Problem*”.

Firmware developer unfamiliar with ease of extraction with physical access.

What went wrong?

2. Insecure Network Services

Never disabled SSH

Never disabled the local web server on the mirror that was used for setup.

Why?

Leftovers from development and testing

Support over ssh maybe

Low friction deployment and ease of use was a requirement

What went wrong?

3. Insecure Ecosystem Interfaces

- No **real** authentication or authorization
- Served over plaintext http
- No input/output sanitizing
- Lots of opportunity for stored XSS in the config and mirror data
- Probably CSRFable?

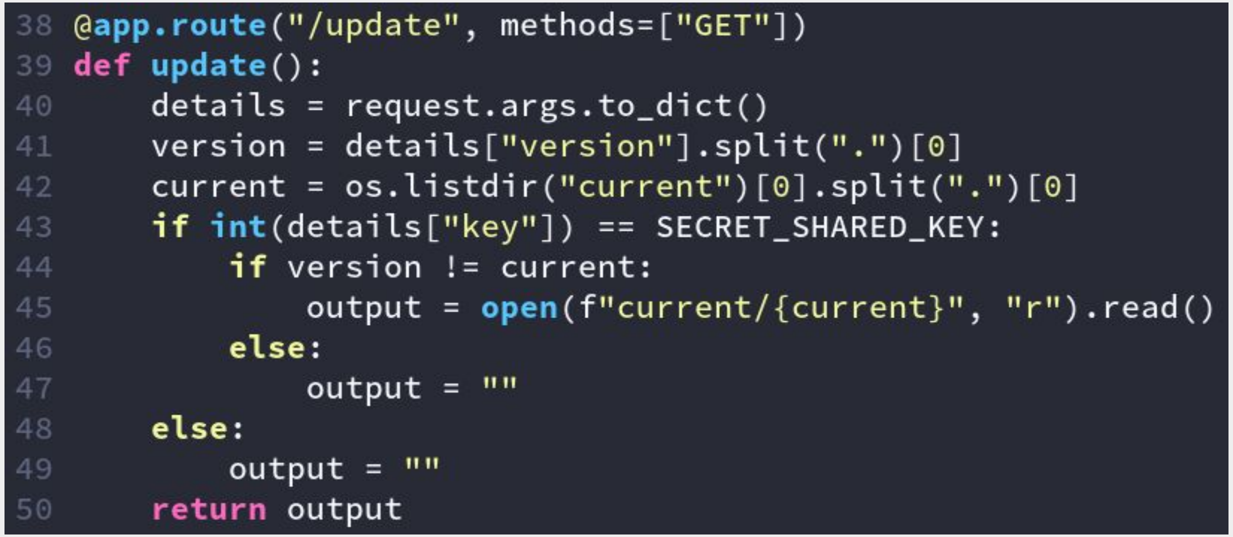
Why? Pace of development, had to make it to market and we went with a technology stack we knew.

Didn’t bother with things like a proper framework, built-in controls or even Let’s Encrypt for encryption.

What went wrong?

4. Lack of Secure Update Mechanism

Let's look at that update function again.



```python
@app.route("/update", methods=["GET"])
def update():
    details = request.args.to_dict()
    version = details["version"].split(".")[0]
    current = os.listdir("current")[0].split(".")[0]
    if int(details["key"]) == SECRET_SHARED_KEY:
        if version != current:
            output = open(f"current/{current}", "r").read()
        else:
            output = ""
    else:
        output = ""
    return output
```

Why? Easy to implement. Solves "*The Warehouse Problem*" really well.

What went wrong?

(Double Jeopardy)

6. Insufficient Privacy Protection & 7. Insecure Data Transfer and Storage

- No HTTPS
- No disk encryption
- Location data and name being stored server-side potentially an issue

Why? Maybe unfamiliar with Let’s Encrypt. Possibly holding on to old notions of crypto performance (even cheap chips have hardware crypto support to some extent now).

Didn’t realize the scope or implications from newer/stricter privacy legislation.

Wouldn’t it have been easy to fix these issues?

“Nothing is more permanent than a temporary solution.”

Root Cause Examination

Potential common root causes for all the issues I experienced

- Rapid pace of development to keep up with the market
- Product requirements
- Low friction deployment & warehouse problem
- Outdated training for hardware and software teams

What can we do?

- Turn-key ecosystems
- Secure base-OS with support for quick and easy updates (docker?)
- Libraries and frameworks to solve problems like updates, first-connection trouble, IAM
- Education and training (IoT Top 10 a good start)

Thanks!
Questions?