Embedded Research & Automation

Hackers to Hackers Conference
2019

brian@security-bits.de ~~ www.security-bits.de ~~ @BadgeWizard

About Me

- Brian
  - @BadgeWizard
  - brian@security-bits.de

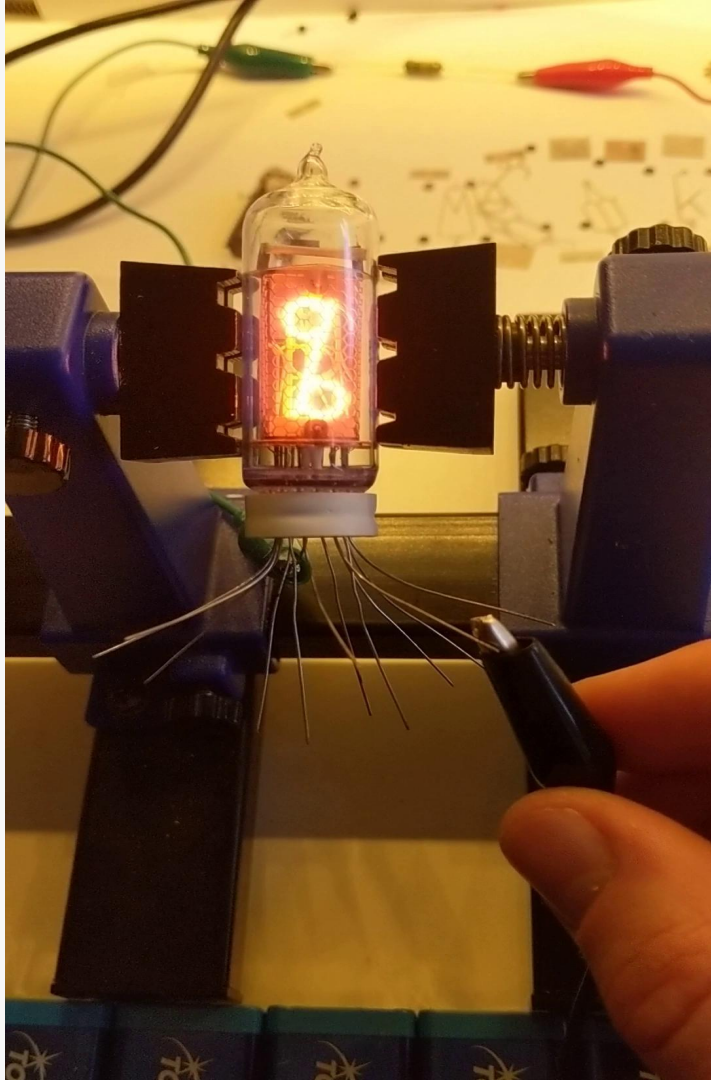
- Security Researcher / Hacker
  - Officially: “Incident Response”
- Hardware-, Embedded-, a bit of Telko-Security

Embedded Testing & Research

1) Dismantle
2) Identify interfaces
3) Reverse circuits
4) Extract data / firmware
5) Perform static analysis
6) Port scan
7) Vuln scan
8) Interface specific assessments
9) Fuzzing

Dynamic Analysis

- Approaches are the same as when testing i.e. a server
- Only the reactions / output are often very different
  - Embedded device might just crash & reboot
  - Or wildly blink and restart the network interface
  - Or just get stuck in a seemingly undefined state
- While the reaction is usually trivial for us to see it can be hard for an autonomous testing platform



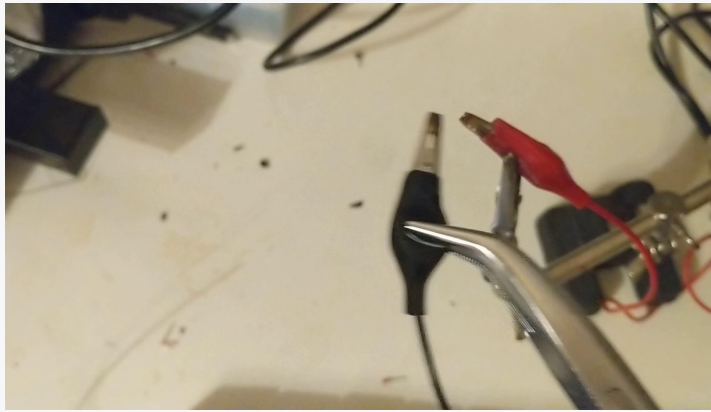
Agenda

- Embedded device outputs
- Creating an interface to collect reactions
- Utilizing sensors
- Invasive and non-invasive approaches

- Automating physical input

Typical Embedded Device Outputs

- LEDs
- Sounds
- Vibrations
- Actions
  - I.e. a coffee machine making coffee or a lock opening
- Power draw

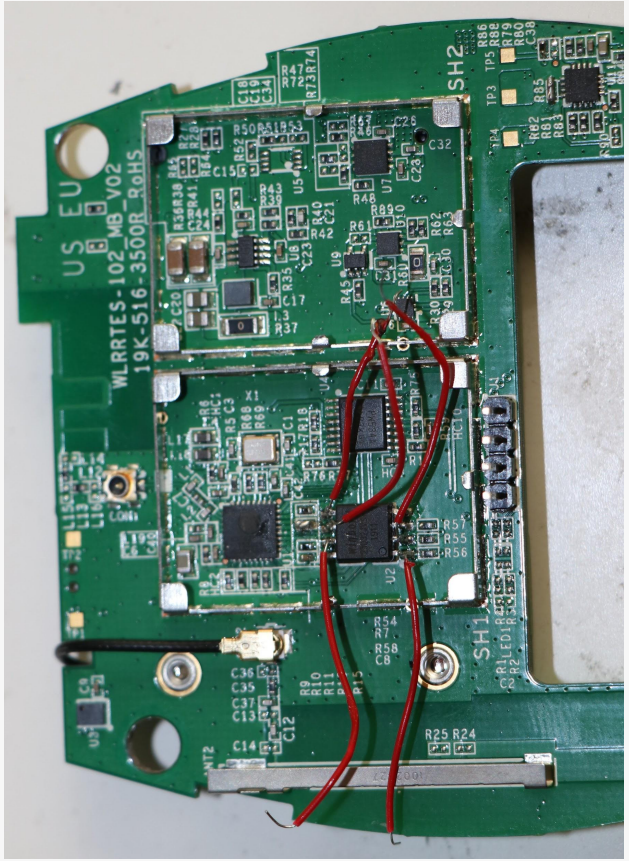


- Blue smoke
  - Though that always is a really bad sign

All Output is Electric

- All components on a device are powered by electricity

- Thus, instead of looking at the output of the component, we look at the voltage driving the component
- Just solder a wire where you need it and utilize that



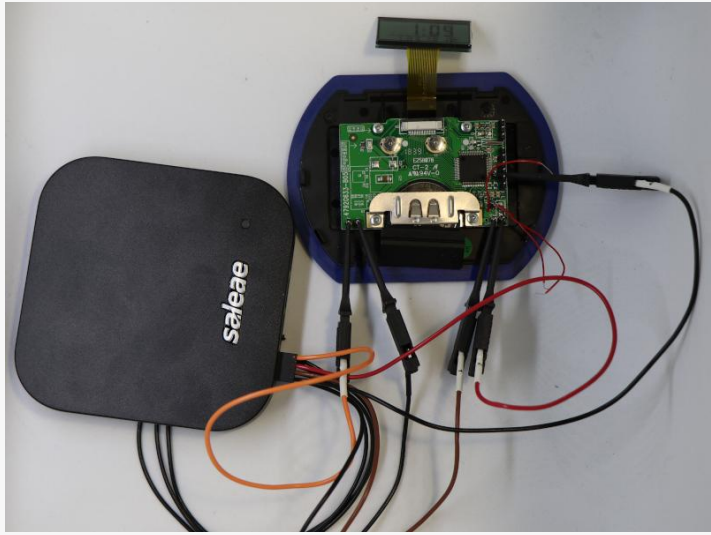
Logic Analyzers

- Logic Analyzers simply measure output voltages and visualize them in a digital manner
  - Low voltages as 0, higher voltages as 1

- Thus we can easily create a trace for signal

Logic Analyzers

- Logic analyzers can cost anything between 10$ (cheap Chinese clones) and multiple thousand (high speed, many channels)
- My personal best experience so far was with Saleae Logic Analyzers
- The very cheap ones often put too much load on the device and thus crash it



Logic Analyzers

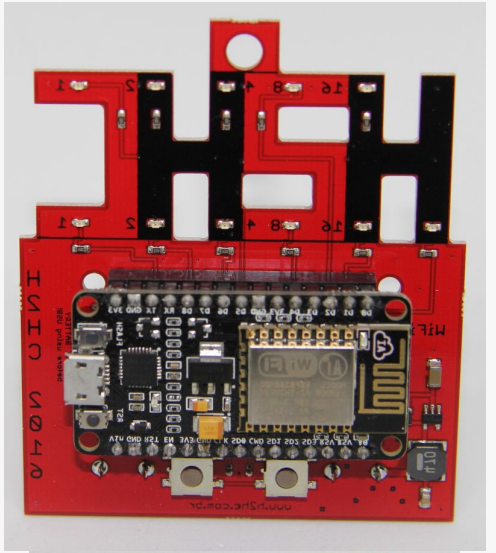
- Most Logic Analysers collect traces for a certain amount of time
  - Start is either triggered by a state change or by hand
- Then they make all the data available
  - I.e. as a CSV

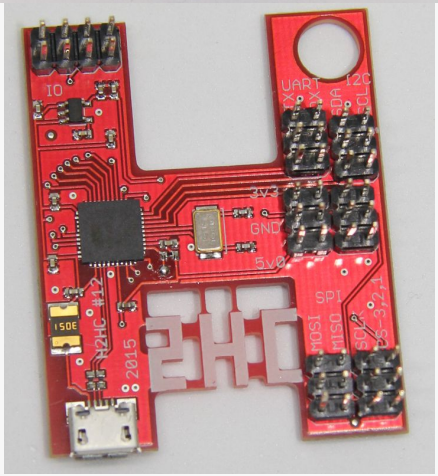
- Gives us the option to later on see what happened but no live view

```
17.038765044000002, 0, 1, 0
17.038765048000002, 0, 0, 0
17.038765052000002, 0, 0, 0
17.038765056000003, 0, 0, 0
17.038765059999999, 0, 1, 0
17.038765064000000, 0, 1, 0
17.038765068000000, 0, 1, 0
17.038765072000000, 0, 1, 0
17.038765076000001, 0, 1, 0
17.038765080000001, 0, 1, 0
17.038765084000001, 0, 1, 1
17.038765088000002, 0, 1, 1
17.038765092000002, 0, 1, 1
17.038765096000002, 0, 1, 0
17.038765100000003, 0, 1, 0
17.038765103999999, 0, 1, 0
17.038765108000000, 0, 1, 1
17.038765112000000, 0, 1, 1
17.038765116000000, 0, 1, 1
17.038765120000001, 0, 1, 1
17.038765124000001, 0, 1, 1
17.038765128000001, 0, 1, 1
17.038765132000002, 0, 1, 1
17.038765136000002, 0, 1, 1
17.038765140000002, 0, 1, 1
17.038765144000003, 0, 1, 1
17.038765148000000, 0, 0, 0
17.038765152000000, 0, 0, 0
17.038765156000000, 0, 0, 0
17.038765160000001, 0, 0, 1
17.038765164000001, 0, 0, 1
```

Dev Boards

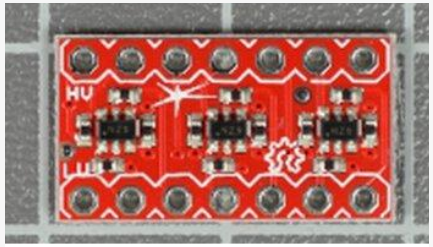
- There is big selection of dev boards on the market
  - Arduino, ESP32, ESP8622, STM boards, random ARM boards
  - Choose whatever you are familiar with
- All of them offer IO pins
  - Which is the only thing you'll need for the start



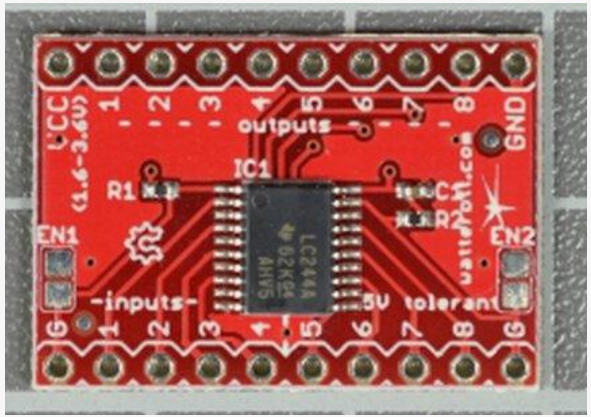


Voltages

- You just need to make sure to work in the correct voltage range
- Dev boards usually run at 3V, 3.3V or 5V
  - With the obvious exceptions
- Target devices will often go down to 1.8V
  - For low power devices and various ARM chips



watterott.com



- Too high voltage might fry your dev board, too low might simply not work

Otherwise you might just want a level shifter

Example: ESP32 & MicroPython

- < 10 lines of code for writing an interrupt handler
- When the signal on pin 32 goes from high to low (IRQ_FALLING) do_magic is called

- Can easily be combined with a serial interface passing the events on to a PC

```
btn1 = Pin(32, Pin.IN)

btn1.irq(trigger=
Pin.IRQ_FALLING, handler=callback)

def callback(p):
  if p==Pin(32):
    do_magic()
```

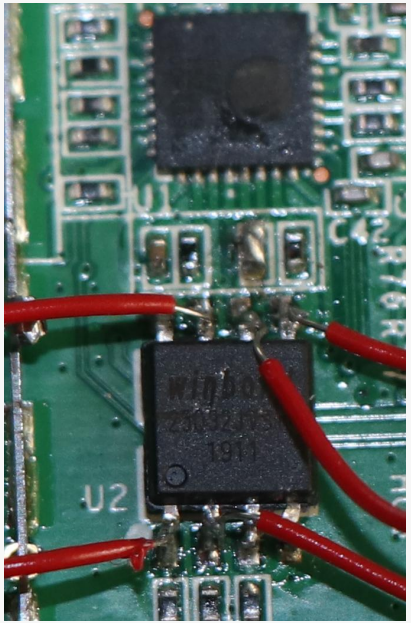
Automation

- Simply extend your fuzzing / testing script by adding pySerial
- Which then reads the output of the interface
- And allows you to log it with whatever attack you have just run

Soldering & Heat Break Stuff

- Everytime you solder it leaves traces on the target device
- Depending on how experienced you are, you will break things
  - Which is just a part of learning

- What do you when aren't allowed to solder? Or open the target device

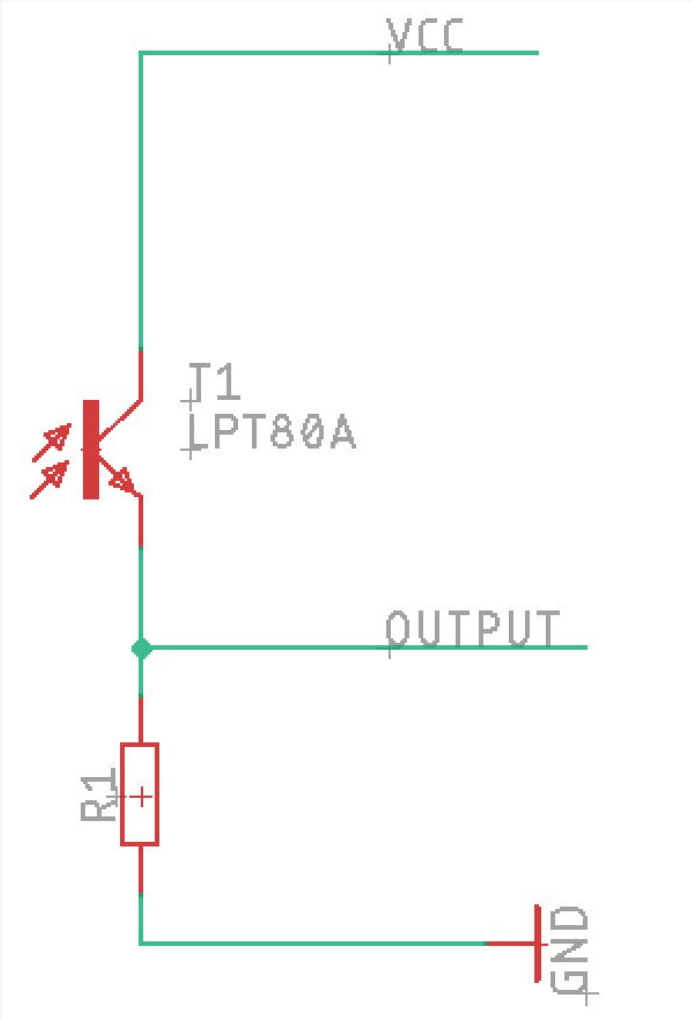


Non-Invasive Approach

- Sensors are your friend
  - Optical: Light Sensors, Photodiodes, Phototransistors
  - Sound: Microphone
  - Vibration: Accelerometer
  - Actions: Switches? IR Light barriers?

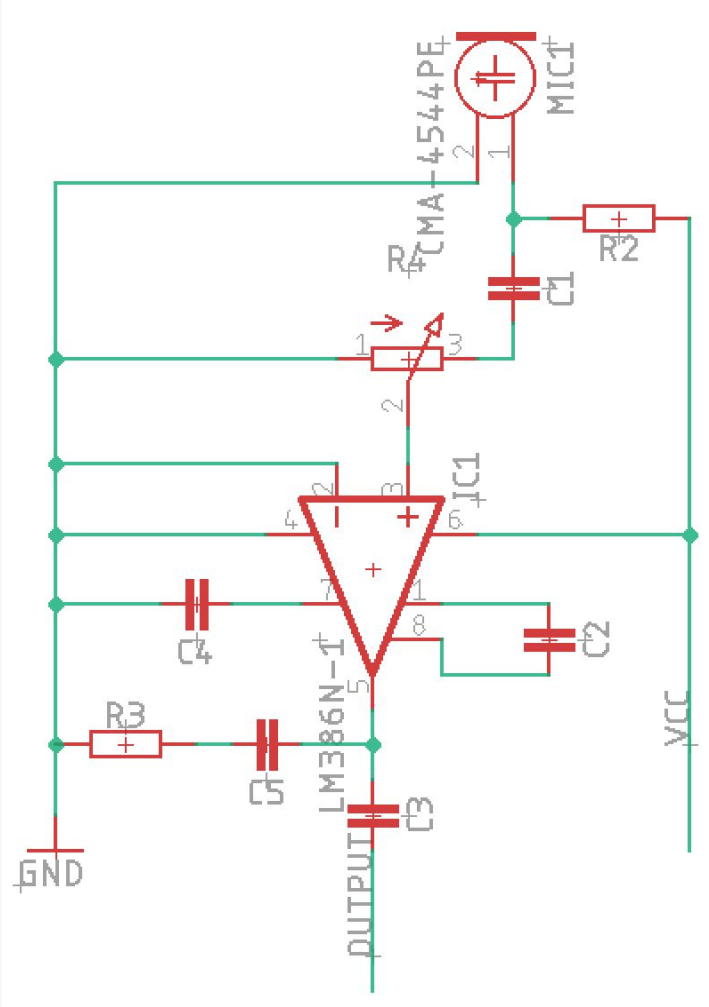
Optical

- A phototransistor is a transistor which is switched by light
- The design on the right is the most simple circuit
  - The Output pin is connected to a microcontroller
  - R1 pulls the line down to GND while the transistor is idling
  - When exposed to light the voltage on the Output line rises
- The output here is rather strong but analog
  - Thus might not be enough to drive a digital pin



Audio

- Signals from microphones are really really weak
  - And have to be amplified
- Thus MIC1 is the actual microphone, R4 is variable resistor for volume / output voltage control and the rest is the amplifier

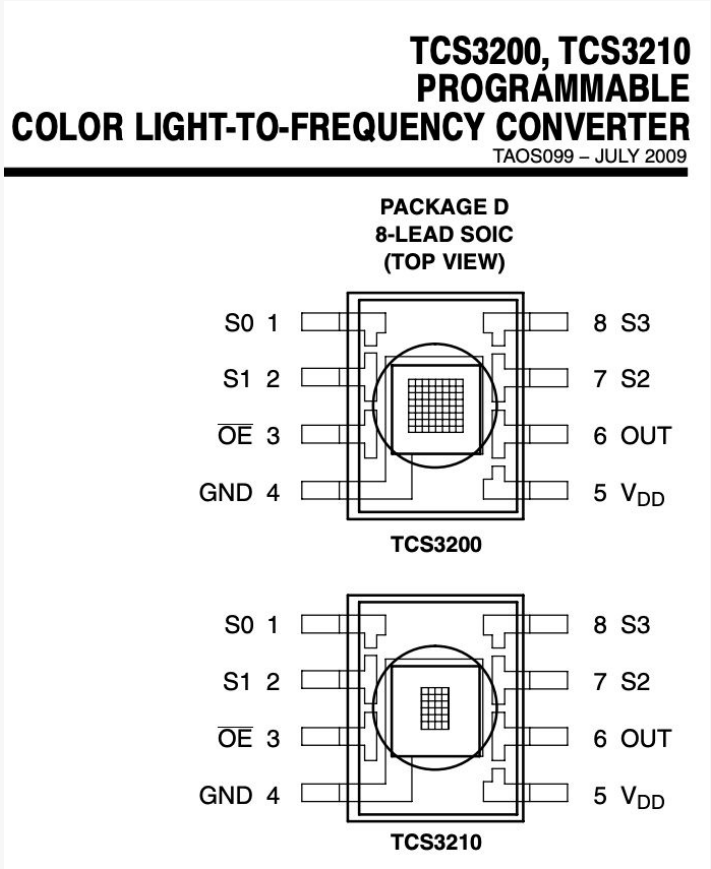


Noise

- Both audio and light are very susceptible to noise!
- The sensors have to placed in such a way that they aren’t exposed to a lot of pollution
  - I.e. cover the light sensor with a bit of tape or place it in a thin paper tube
  - Cover the microphone with some foam

Color

- Sometimes on/off isn't enough
- Especially when working with LCD displays you might need a color
  - Red background for an error, otherwise white
  - Or a yellow triangle as a warning sign
- TCS3200 is a cheap color sensor that outputs detected color as specific frequencies
  - Can be directly controlled from a microcontroller



Actions - Be Creative

ESE-13V01A
Digikey

D2F-L-A1
Digikey

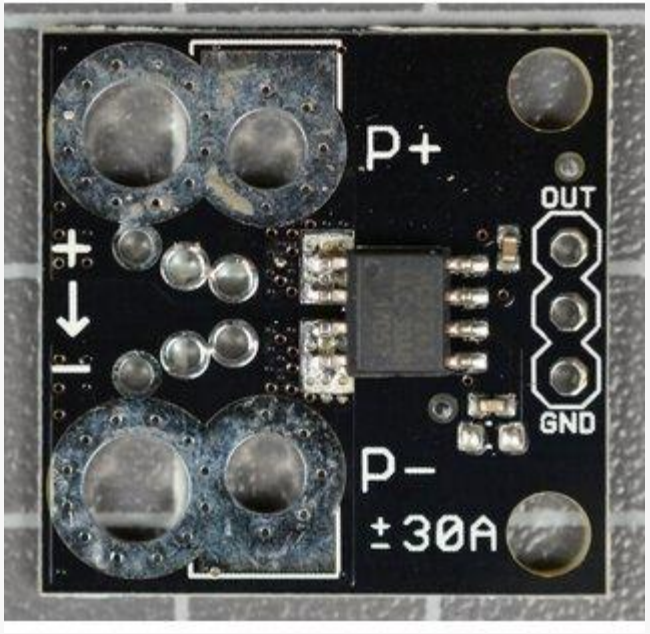
TRCT 5000
Digikey

SEN11050
Sparkfun

Google is your friend :)

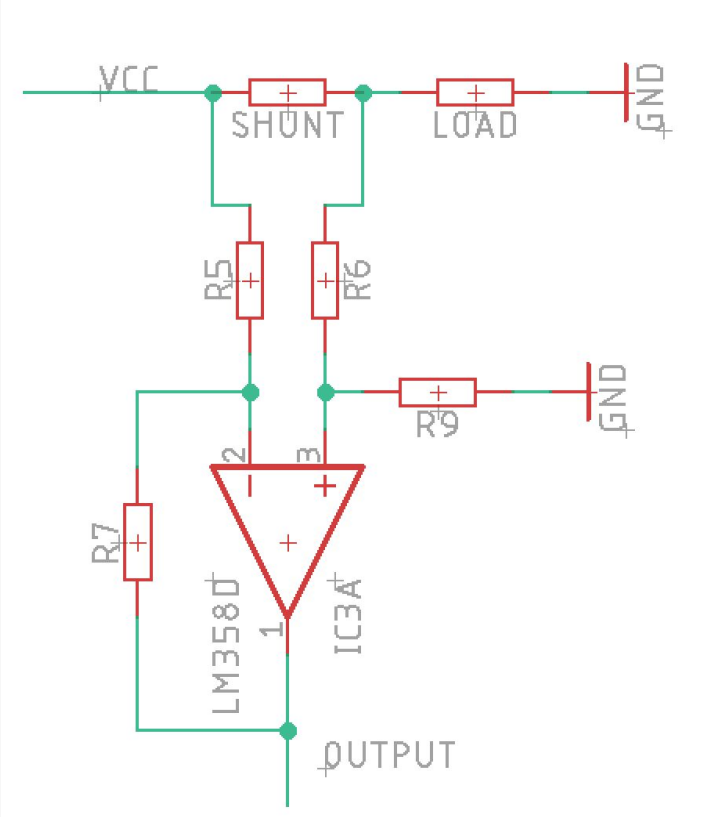
Power Consumption

- Can be key to understanding and fingerprinting in which state a device is
  - More to do -> more load -> consumption
- Current sensor
  - I.e. ACS725
  - Able to measure up to 10A
  - Available for both AC and DC



Power Consumption

- Shunt-Resistor with Op-Amp
- Utilizing Ohm's law
  - U = I * R
  - A current running through a resistor results in a specific voltage over the resistor
  - Resistor should be very small not to influence the circuit
  - Thus the amplifier
- Output can then be read by an analog pin on the micro controller



Device States

- Device states can be characterized by simple properties
  - Power Consumption
  - (Dynamic) State of certain outputs
- When putting together a test rig, one simply has to perform a little bit of fingerprinting

- When does which LED turn on / off?
- Do LEDs blink?
- How often do they blink?
- How quickly do they blink?
- How does the power consumption fluctuate i.e. when WiFi on a device comes up?
- When does the device beep?

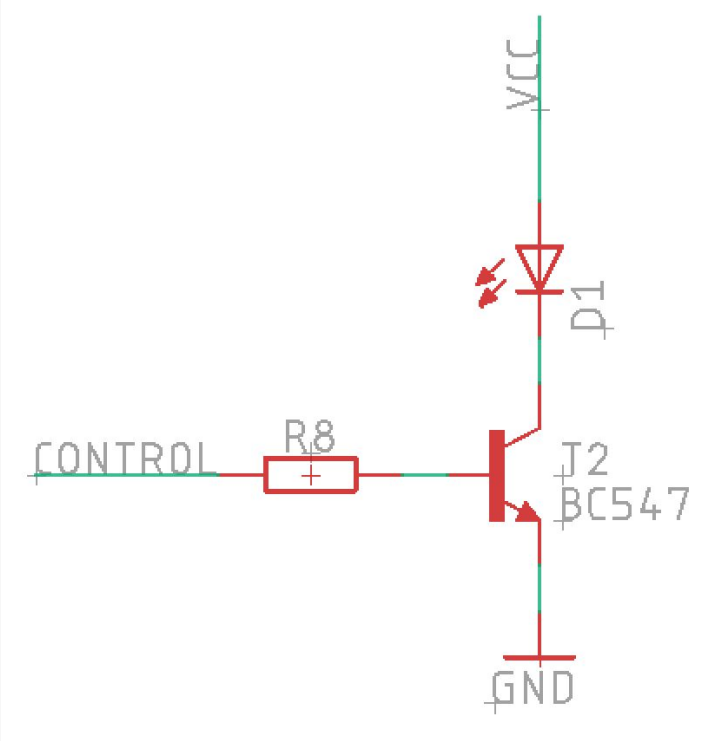
Forcing States

- While adding something to the power lines we might just as well add a relay
  - Which we can use to restart the device
- When fingerprinting the device we could also time the boot process
  - I.e. start device, wait for 2 seconds, fuzz for 5 seconds (while the device is in the bootloader)
  - Then force a reset

Creating Input

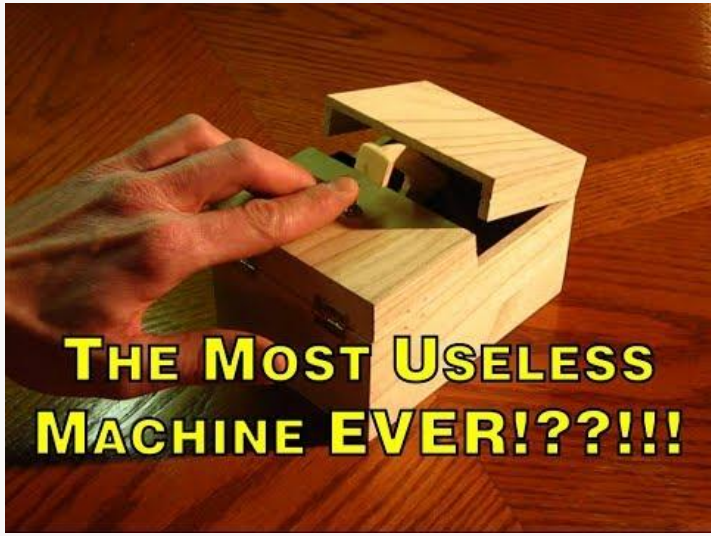
- Input, yet again, is electric

- Transistors or relays can be used to pull lines to certain levels



Creating Input

- Movement can be created by using servo motors
  - Or by taping the target device to a fan and letting it turn a little
- Touch input can also be created with a motor
  - Just make sure that the tip is capacitive if necessary



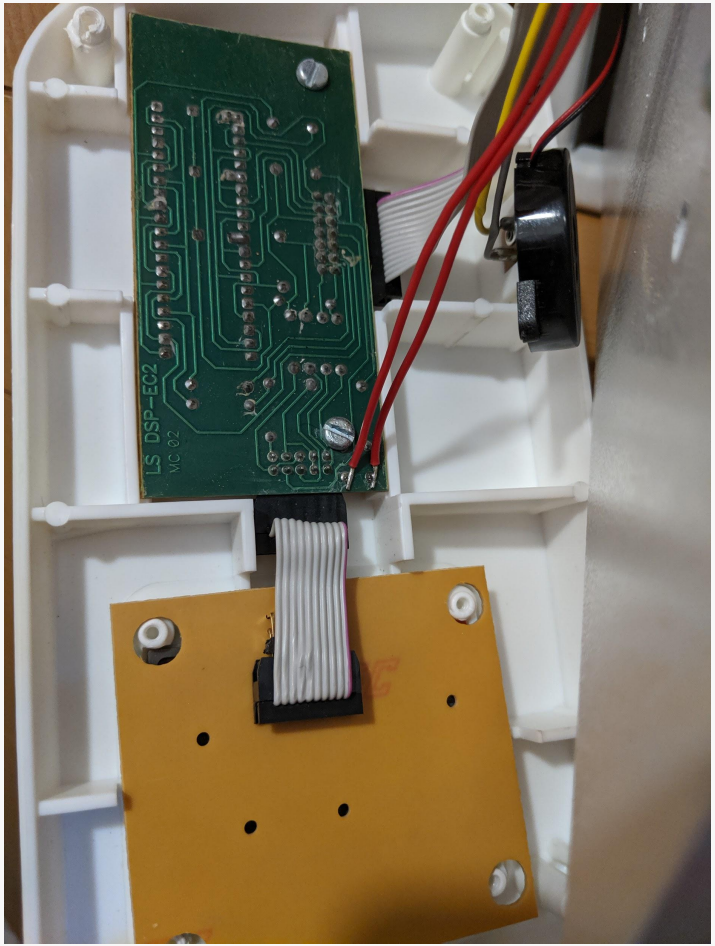
Safety first

- Don’t just play with devices that are attached to the mains
- Electric shocks really suck
  - Trust me I’ve had a few
  - They might not generally be deadly but that doesn’t make things any better

- If in doubt, find a nice person to show / teach you

Creating a Setup

1. Identify all I/O you want to hook into
2. Select appropriate ways of attaching
    a. Don’t think too complicated, there are sensors for next to everything!!!
3. If invasive, check the supply voltages
4. Make sure your cables are long enough
    a. Having a free-flying-tape setup might look cool but it will sooner or later create problems



Putting Things Together

- First approach usually is a breadboard
  - Which it really should be, to ensure that the setup actually works
- Most parts come in “THT” packages
  - Wired and can be directly plugged into a breadboard
- Thanks to the maker scene most other strange parts like sensors come as break-out boards
  - They can be plugged into the breadboard or connected with wires

Putting Things Together

- Add one component after the other
  - Always write a few lines of code to ensure that it does what you expect
- It’s not fun if your measurements don’t work because the trigger you went for does not work
- Fixate sensors with a little tape or Blu-Tack so that they don’t move while working

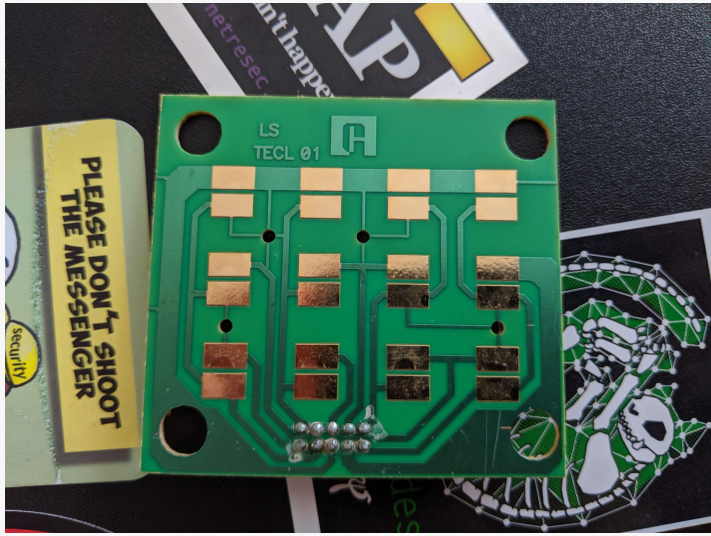
Attack Surface & Interfaces

LED Display
4 Digits + a little extra

12 Buttons
0-9, Lock Func

A Little Bit Invasive

- We'd have to cut out the pin pad
  - Or use actors to press the buttons
- We need 12 transistors / relays to control the input
  - Soldering something to the metal surfaces of the buttons is easy
  - But we might as well use the header
- Connect everything to an ESP32
  - Basically done!



Understanding the Safe

- When the PIN is correct
  - The safe opens
- When the PIN isn't correct
  - Sometimes nothing happens
    - Glitch?
  - We just need to press the Func button once

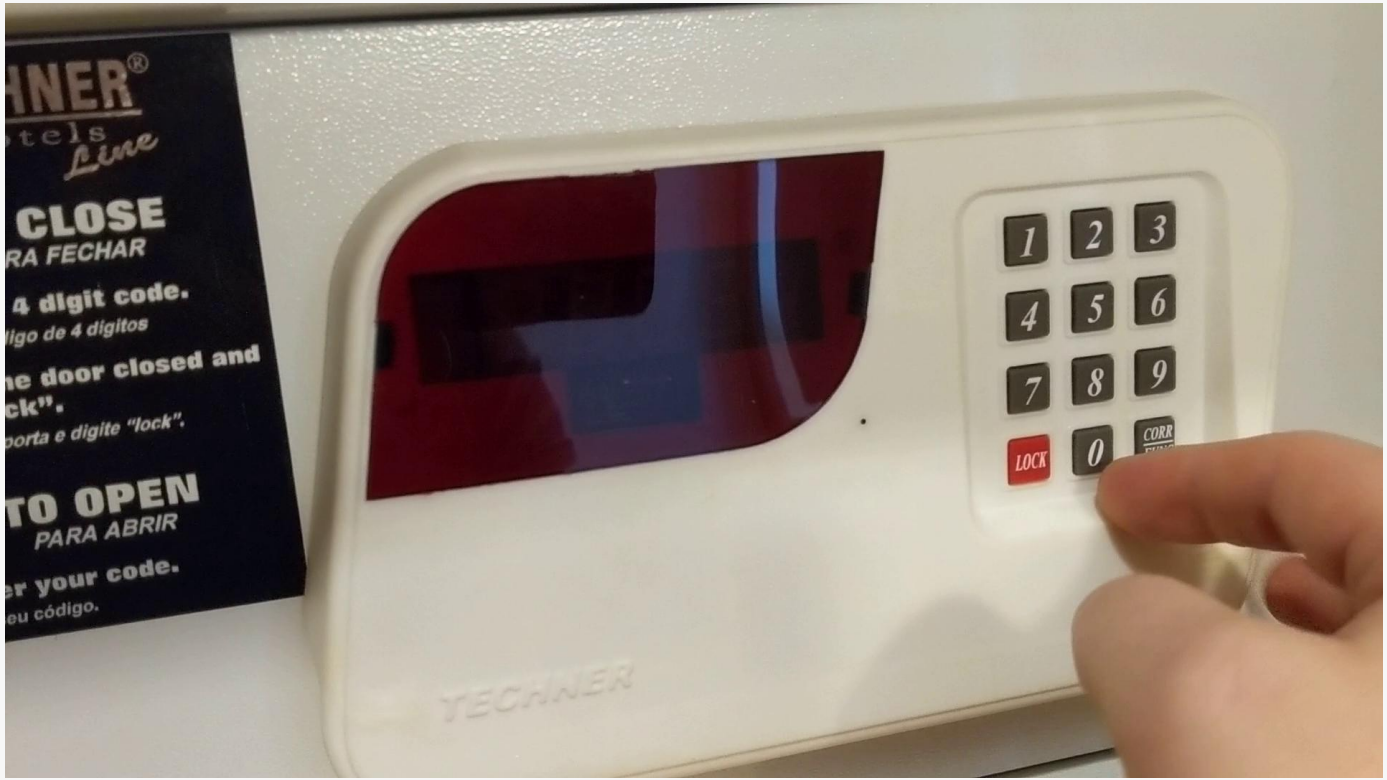
A Little Bit of Code

- Code on the right plus a loop is everything we need for bruteforce

- Define the IO pins
- Pull high
- Wait
  - The wait has to be adjusted
- Drop to low
- Hope for a callback

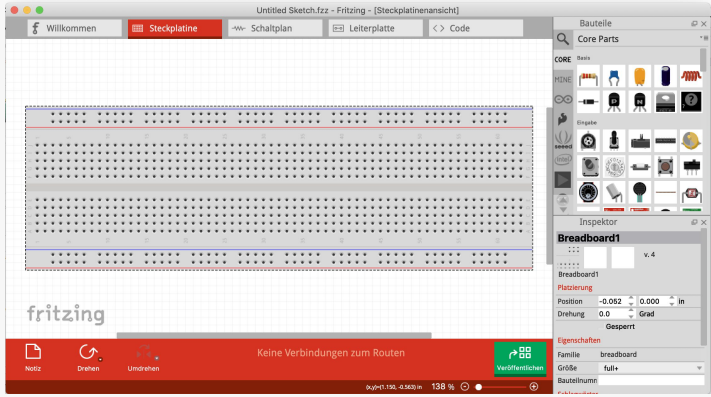
```
btns = [btn1 = machine.Pin(2,
machine.Pin.OUT, machine.Pin(4,
machine.Pin.OUT), ...]
# etc

def p_btn(id):
  btns[id].value(1)
  time.sleep_ms(250)
  btn[id].value(0)

dispx = Pin(32, Pin.IN)

dispx.irq(trigger=
Pin.IRQ_FALLING, handler=callback)

def callback(p):
  if p==Pin(32):
    ##Open! :)
```

Funnily, no Bruteforce Necessary



Scaling & Reusing

- When doing work like this regularly you'll want a stable but flexible setup
- Then it's time to actually design a custom PCB
  - And have it manufactured
- This also ensures that you can reuse your codebase



Invasive vs. Non-Invasive

- Invasive approach is usually easier
  - Allows you access to BUS communication (which I haven't discussed)
  - More stable towards noise
  - By far less tape & BluTack
- Non-Invasive approach is always an option
  - All devices "leak" very helpful information
  - It might just require some more creativity

Thanks for your time

Questions?

brian@security-bits.de ~~ www.security-bits.de ~~ @BadgeWizard