Beyond HTTP: A Crash Course on Hacking IOT's Non-HTTP Attack Surface

id -Z @almostjson

Jesson Soto Ventura
(@almostjson)
Senior Security Consultant
Carve Systems

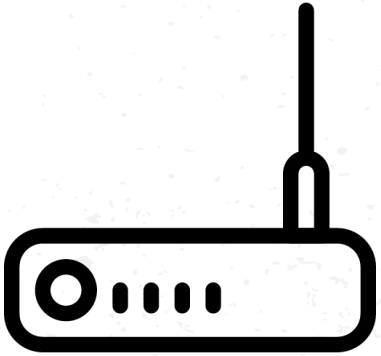
5
years

50
IOTs Hacked

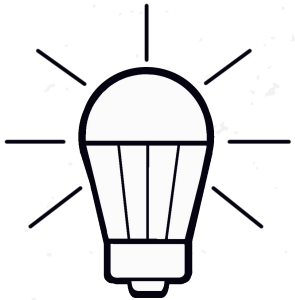
Let’s go back to 2017

date -s ‘06/01/2017’

NYC
NEW YORK CITY

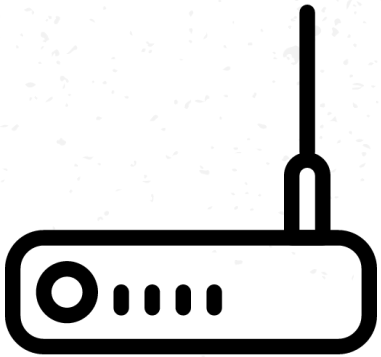


Internet Hotspot



Industrial Smart Light

date -s ‘06/01/2017’

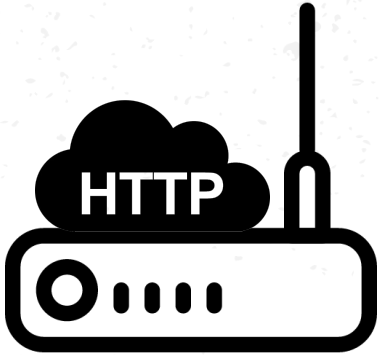


Internet Hotspot

IOT Security Spectrum

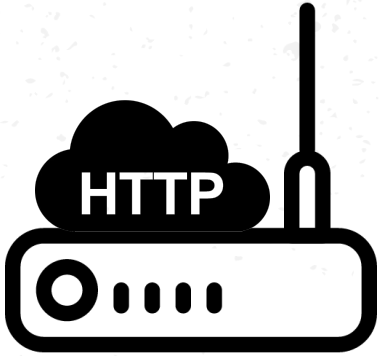
Industrial Smart Light

cat hotspot_root.py



1. Login to web interfaces
2. Go to diagnostics menu
3. Find ping
4. ;reboot; & `reboot` | ${reboot}
5. Submit

cat hotspot_root.py



1. Login to web interfaces
2. Go to diagnostics menu
3. Find ping
4. ;reboot; & `reboot` | ${reboot}
5. Submit

Steps 3,4 may vary but often the web interfaces is exploitable

date -s ‘06/01/2017’

Internet Hotspot

IOT Security Spectrum

Industrial Smart Light

cat light_root.py

cat: light_root.py: Permission denied

date -s ‘06/01/2017’

Internet Hotspot

IOT Security Spectrum

Industrial Smart Light

mkdir beyond_http

ls -l beyond_http

1 Network Layer

Bluetooth 2

3 Android Apps

cd ‘Project Prometheus’

cat prometheus.md | grep ‘background’

- ➔ A few members of the Carve team bought smart grills and our goal was simply to hack them and start a fire.
- ➔ What follows is all a reenactment based on a true story

cat prometheus.md | grep attack_surface

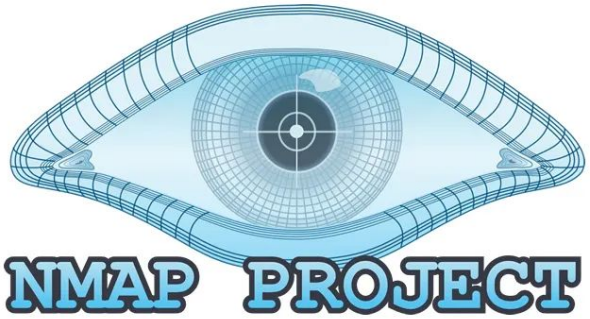
- ➔ The grill's attack surfaces included: iOS, Android, WiFi, Bluetooth
- ➔ So where do we start? With a port scan!

cd network/

cat network.md

- ➔ The network layer is often the primary communication layer for IOT devices. They are “internet” connected devices after all!
- ➔ Yes, the web interface is the most common communication channel, but other channels can exist. We should find them and know about them.

cat network.md | grep ‘getting started’



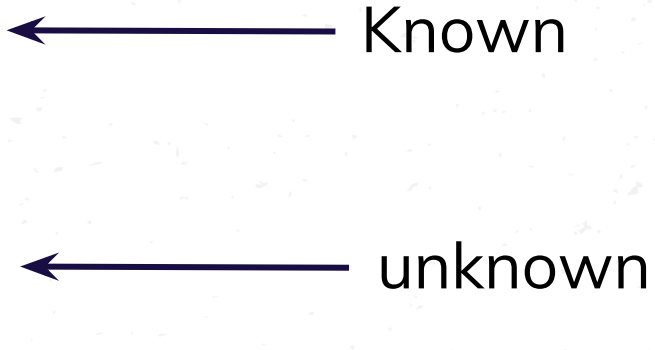
nmap.com

1. Scan the device for open ports:
   nmap -sC -sV -O -p- x.x.x.x/24
2. Review the output to determine, if the open ports are useful or if you should just skip them

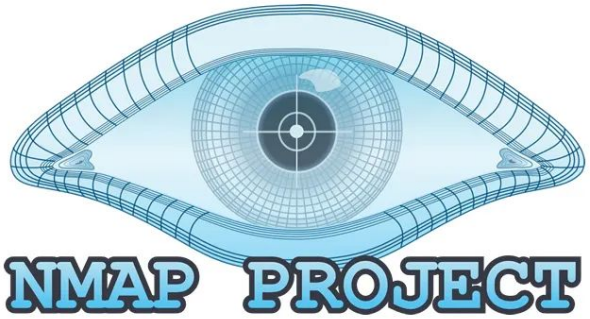
This only scans tcp, but you probably only care about tcp ports anyways

cat network.md | grep ‘sample’

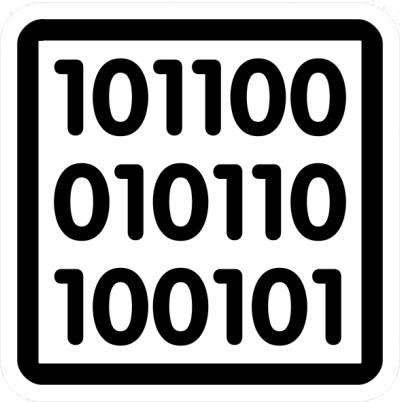
```
nmap -p- 192.168.3.1
Starting Nmap 7.92 ( https://nmap.org ) at 2022-06-03 11:08
EDT
Host is up (0.0037s latency).
PORT     STATE   SERVICE
53/tcp   open    domain
80/tcp   open    http
443/tcp  open    https

4420/tcp  open    unknown
4282/tcp  open    unknown
```



cat network.md | grep ‘useful ports’

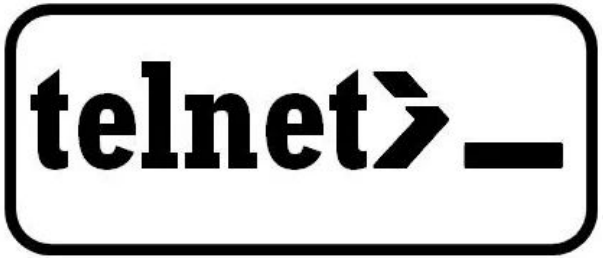


Known:
Nmap is able to tell you what the service is an when you check, you agree its the same service



Unknown:
Nmap is unable to tell you what the service or you disagree with the service it identified

cat network.md | grep ‘known ports’



cat network.md | grep ‘known’ | grep ‘shells’

- ➔ Highly likely to give you a shell!
- ➔ Also highly likely to require authentication

Tips for bypassing authentication:

- Default passwords!
- REing the mobile apps

23

22

5555

cat network.md | grep ‘known’ | grep ‘others’

- ➔ Use your favorite search engine to look up the protocol:
  - <service> tools
  - exploits <service>
  - <service>/<port> usage
- ➔ You want to understand enough of the service and its potentials. Maybe you can creatively use it to gain additional information.
- ➔ Maybe it’s just useless.

SNMP

?

cat network.md | grep ‘unknown ports’

- ➔ Send some data to the unknown port:
  - Make an HTTP request
  - Send an empty request
  - Send random data
  - Send “help”
- ➔ Search online for details:
  - Brand + port
  - Model + port
  - Pray to the duck gods
- ➔ RE the mobile application

Highly likely to be a time sink!

cd ‘Project Prometheus’

cat prometheus.md | grep ‘network’

```
nmap -sC -sV -O -p- 192.168.1.100
Nmap scan report for 192.168.1.100
PORT     STATE   SERVICE    VERSION

4577/tcp  open    ssl/unknown
```

unknown

Next step: Try to connect?

cat prometheus.md | grep ‘network’

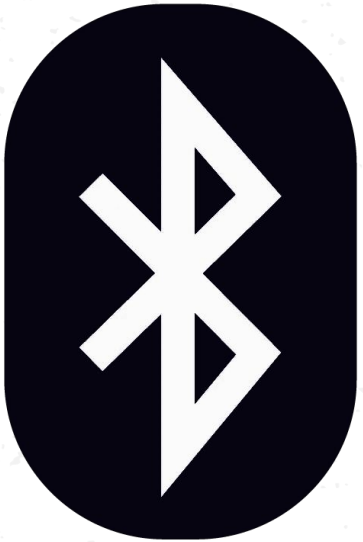
```
curl https://192.168.3.1:4577
curl: (35) error:14094410:SSL
routines:ssl3_read_bytes:s
slv3 alert handshake
failure
```

This error indicates that the endpoint is using mutual TLS to authenticate. Unfortunately, we do not have the key to access the endpoint

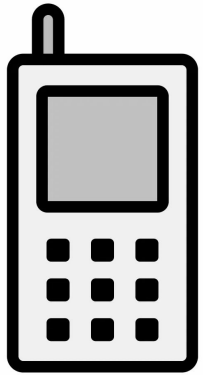
cd ../bluetooth

head bluetooth.md

- ➔ Probably the second most common wireless protocol used by IOT devices
- ➔ Bluetooth interfaces tend to expose a significant amount of functionality

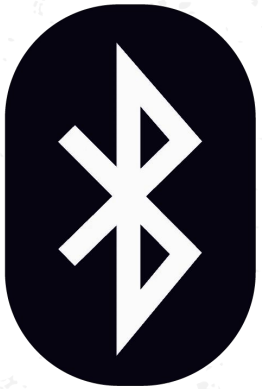


man bluetooth



Client

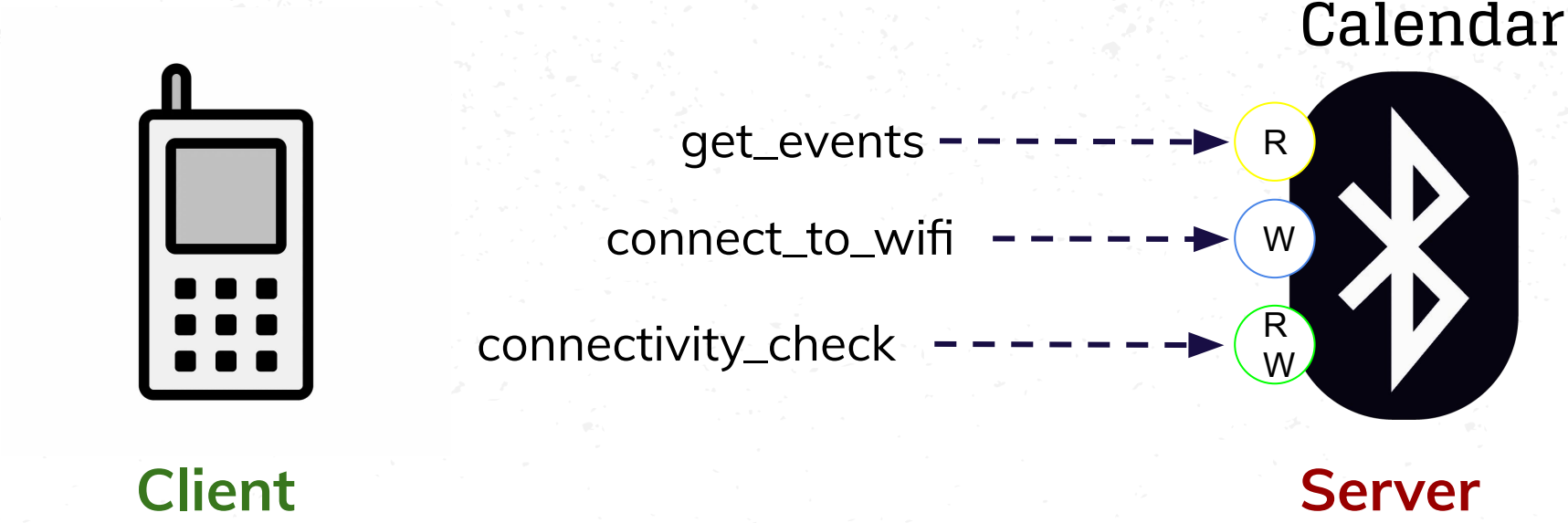
Smart Calendar



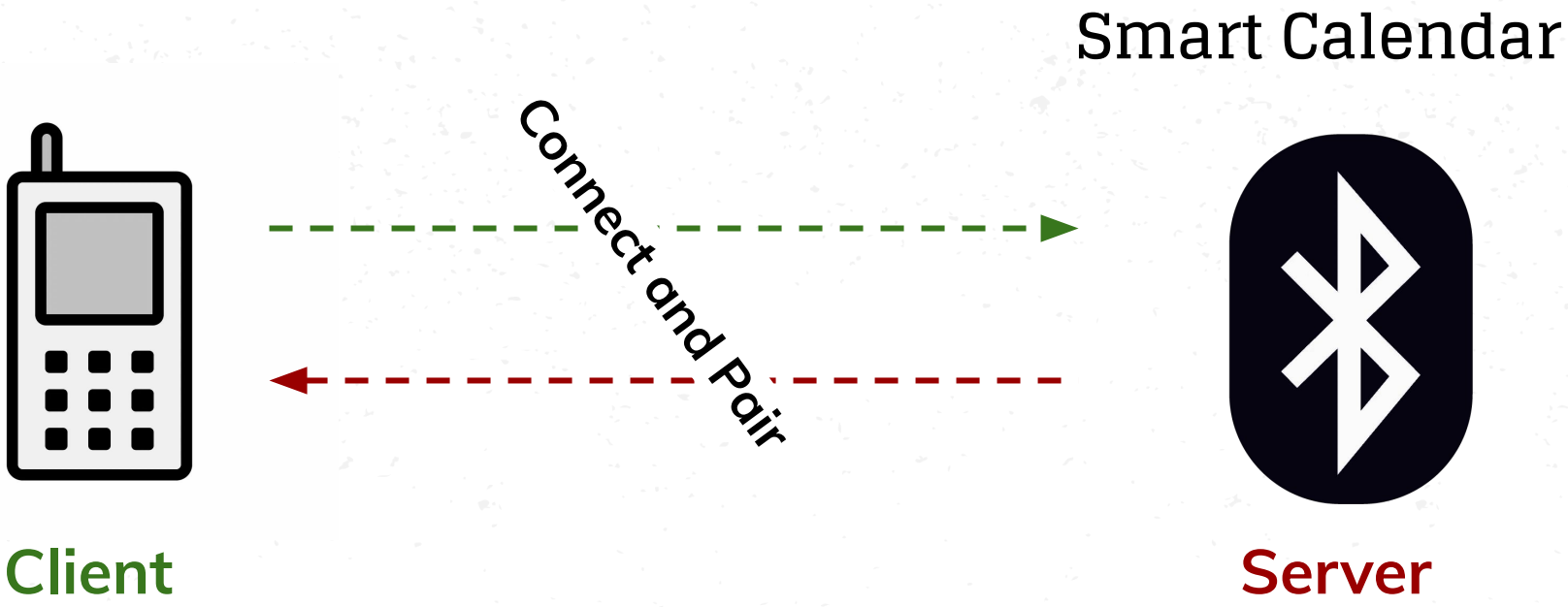
Server

man bluetooth

Characteristics:
An endpoint to access some data



man bluetooth



man bluetooth

man bluetooth | grep auth | grep modes

- ➔ Just works: It just worked out of the box no pairing needed

Authenticated

- ➔ Passkey entry: You had to enter static some passcode
- ➔ Numeric Comparison: You enter a dynamically generated passcode that appeared on some display
- ➔ OOB: Some other pairing method (typically NFC)

man bluetooth | grep auth

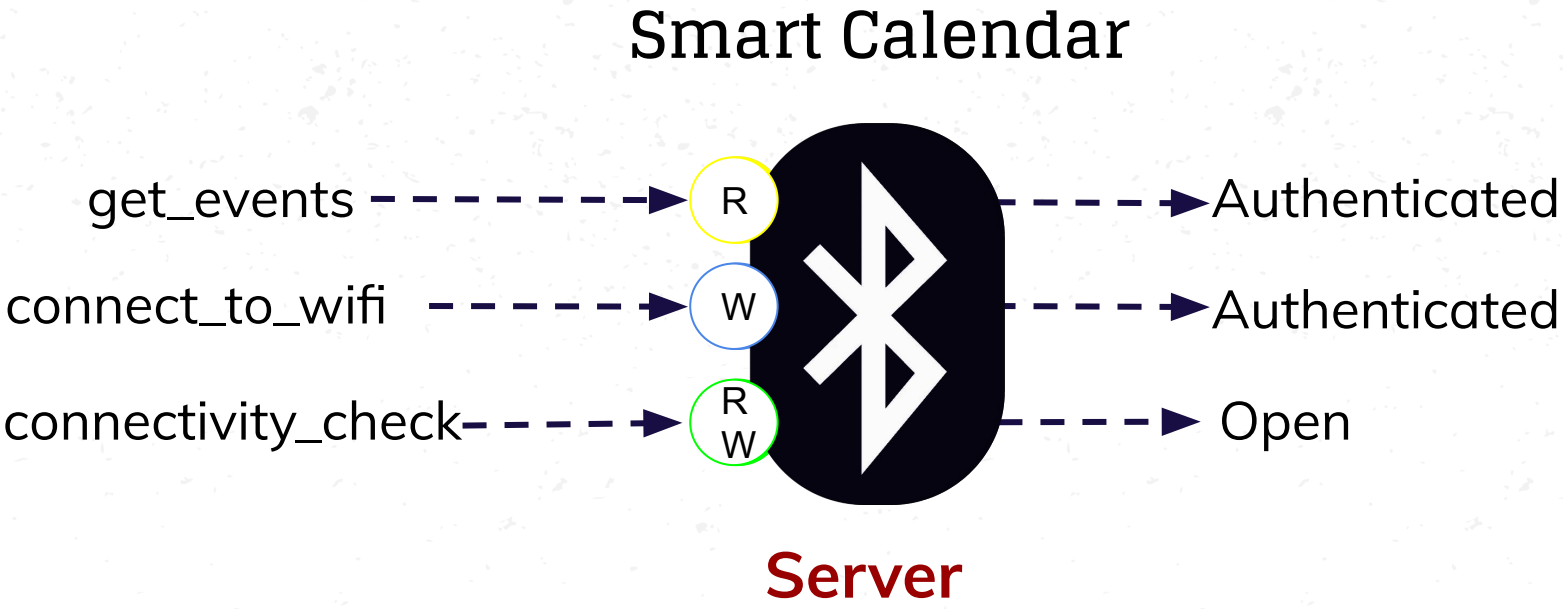
Connecting:
An unauthenticated session

Just Works

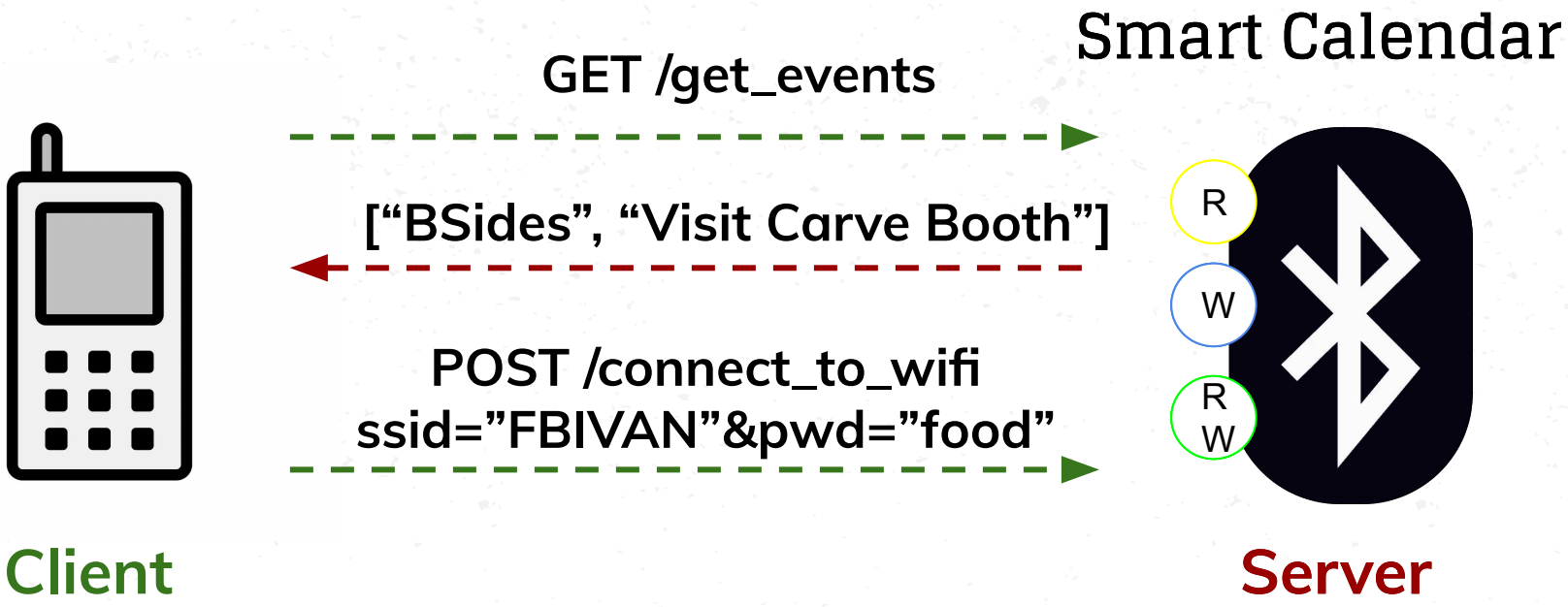
Pairing:
An authenticated session

OOB, Passkey Entry,
Numeric Comparison

man bluetooth

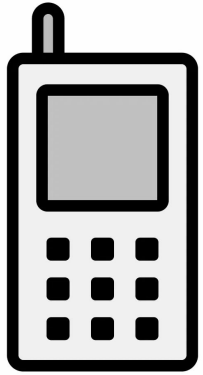


man bluetooth

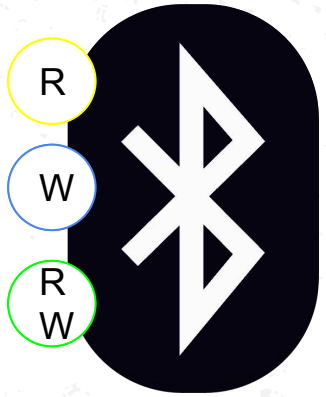


man bluetooth

Smart Calendar



POST /connectivity_check
data="`reboot`"

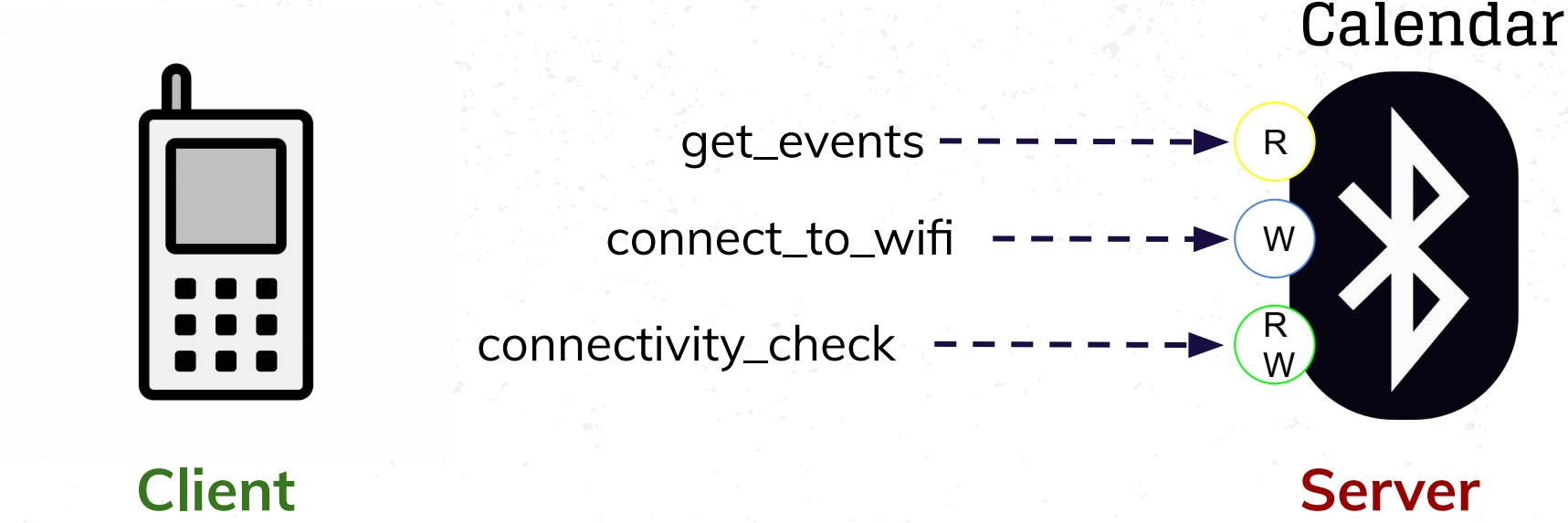


Client

Server

man bluetooth

Characteristics:
An endpoint to access some data



cat bluetooth.md | grep ‘getting started’

https://www.bettercap.org

LightBlue® — Bluetooth Low Energy

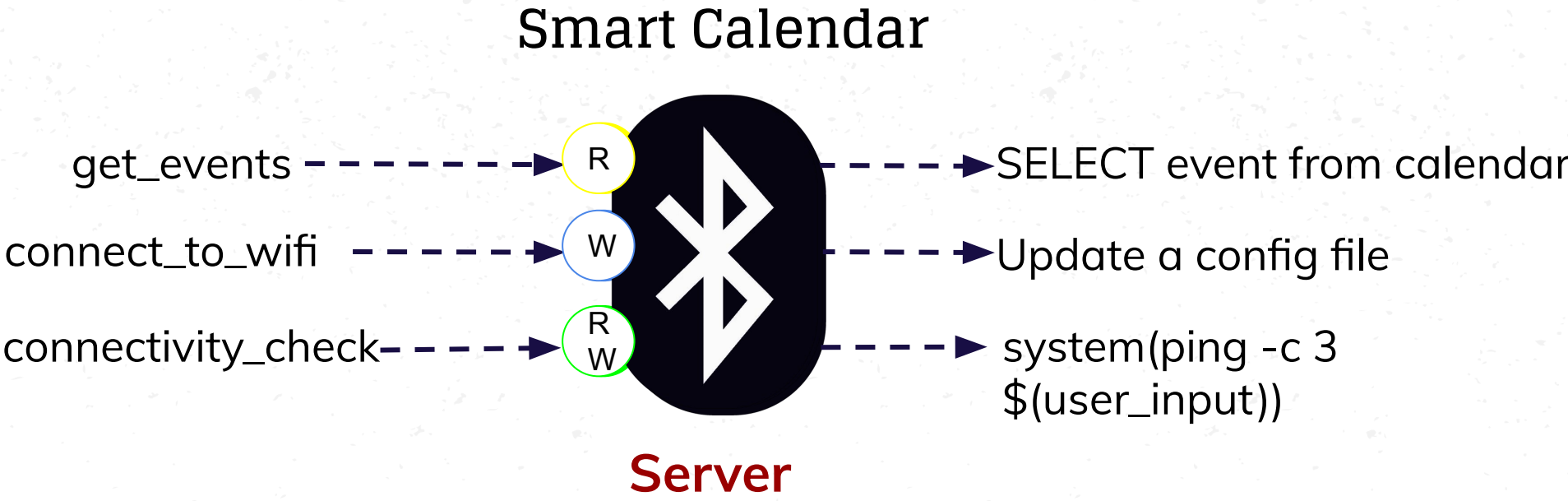
https://github.com/jnross/Bluetility

cat bluetooth.md | grep ‘Using bettercap’

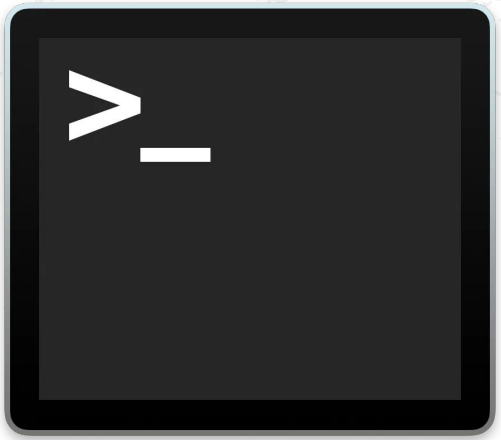
https://www.bettercap.org

- ➔ Lets you scan for bluetooth devices
- ➔ Lets you connect to them (“just works”) and enumerates the entire attack surface to help you understand what’s accessible unauthenticated
- ➔ All in an intuitive TUI
- ➔ Con: Linux only

man bluetooth



cat bluetooth.md | grep ‘RCE’



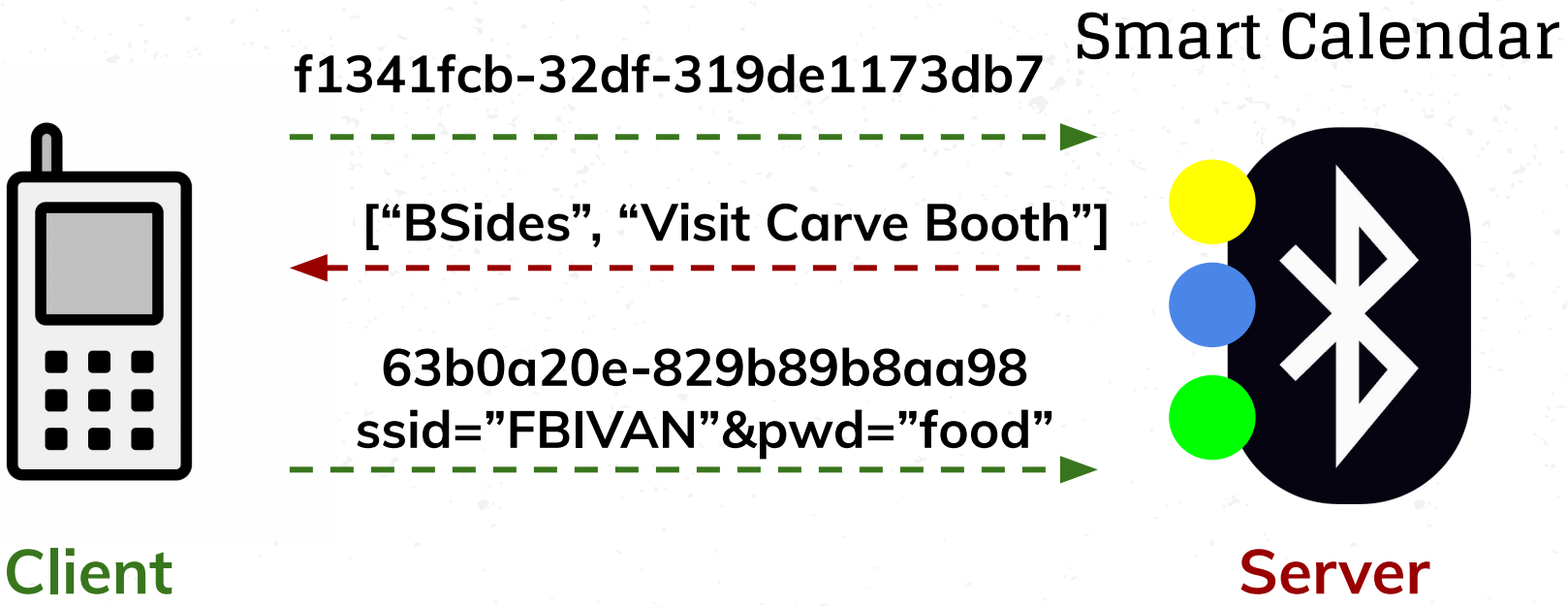
RCE

- ➔ Bluetooth endpoints process data and if you’ve ever looked at any IOT device, you’ll know that data validation isn’t a strength of many IOT devices
- ➔ As such, anywhere you can send strings might be a good place to try out some command injection payloads

cat bluetooth.md | grep ‘Using bettercap’

Handles	Service > Characteristics	Properties	Data
0001 -> 0008	Generic Attribute (1801)		
0003	Service Changed (2a05)	INDICATE	
0009 -> 000d	Generic Access (1800)		
000b	Device Name (2a00)	READ, WRITE	Smart Grill
000d	Appearance (2a01)	READ	Black
000e -> 0012	Device Information (180a)		
0010	Manufacturer Name String (2a29)	READ	Grill Company
0012	System ID (2a23)	READ	000000000
0013 -> ffff	163b0a20e-829b89b8aa98		
0015	1163b0a20e-829b89b8aa98111111	READ, WRITE	insufficient authentication
0018	1163b0a20e-829b89b8aa98222222	READ, WRITE	{}

man bluetooth



cat bluetooth.md | grep ‘hidden’

Hidden Functionality

- ➔ Bluetooth is a great place to hide/include hidden functionality! Remember, the characteristics are long alphanumeric strings, so they must be hard to guess (wrong!)
- ➔ They are wrong! If a device has a bluetooth interface, then it probably has a mobile app!
- ➔ So use the mobile app to figure out what the endpoints do, then figure out what’s not documented

cd ‘Project Prometheus’

cat prometheus.md | grep ‘bluetooth’

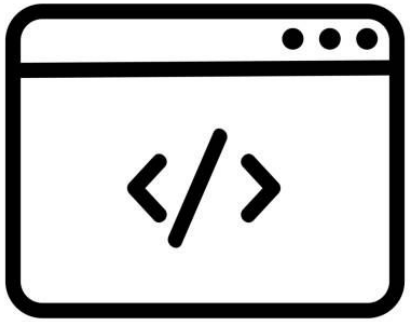
Handles	Service > Characteristics	Properties	Data
0001 -> 0008	Generic Attribute (1801)		
0003	Service Changed (2a05)	INDICATE	
0009 -> 000d	Generic Access (1800)		
000b	Device Name (2a00)	READ, WRITE	Smart Grill
000d	Appearance (2a01)	READ	Black
000e -> 0012	Device Information (180a)		
0010	Manufacturer Name String (2a29)	READ	Grill Company
0012	System ID (2a23)	READ	000000000
0013 -> ffff	163b0a20e-829b89b8aa98		
0015	1163b0a20e-829b89b8aa98111111	READ, WRITE	insufficient authentication
0018	1163b0a20e-829b89b8aa98222222	READ, WRITE	{}

cat prometheus.md | grep ‘bluetooth’

- ➔ Only two endpoints? That’s actually pretty odd.
- ➔ Interestingly enough though, one does not require authentication.
- ➔ Digging further, we see that both are readable and writable but despite being readable they just return empty json strings
- ➔ Wonder, if we can trigger some errors?
- ➔ No, luck! It’s still just a black box.

cd ../mobile_apps

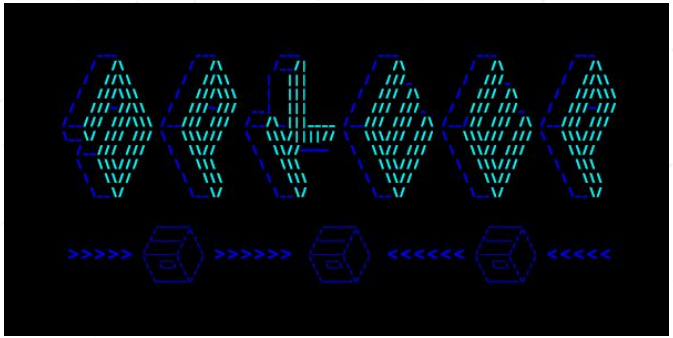
cat mobile_apps.md | grep ‘information’



Reference
Implementations

cat mobile_apps.md | grep ‘getting started’

Downloading an APK



https://github.com/EFForg/apkeep

jadx-gui app.apk

https://github.com/skylot/jadx

- ➔ Jadx takes your APK file and converts it into essentially Java source code (technically pseudo code, but it's basically java)
- ➔ With access to the Java pseudo code, you can:
  - Find hard coded secrets
  - Review implementation details about protocols
  - Figure out how the device works
  - Find hidden API routes

cd ‘Project Prometheus’

cat prometheus.md | grep ‘java review’

https://github.com/skylot/jadx

➔ Based on our previous work, we’re looking for the following in the source code:

- Can we figure out what the characteristics are for?
- Can we figure out how to interact with them?
- Can we get a copy of the mTLS certificate needed to talk to port 4577?
- If we can, what APIs should we know?
- What do we not know that we should know?

cat prometheus.md | grep ‘bluetooth’

Handles	Service > Characteristics	Properties	Data
0001 -> 0008	Generic Attribute (1801)		
0003	Service Changed (2a05)	INDICATE	
0009 -> 000d	Generic Access (1800)		
000b	Device Name (2a00)	READ, WRITE	Smart Grill
000d	Appearance (2a01)	READ	Black
000e -> 0012	Device Information (180a)		
0010	Manufacturer Name String (2a29)	READ	Grill Company
0012	System ID (2a23)	READ	000000000
0013 -> ffff	163b0a20e-829b89b8aa98		
0015	1163b0a20e-829b89b8aa98111111	READ, WRITE	insufficient authentication
0018	1163b0a20e-829b89b8aa98222222	READ, WRITE	{}

cat prometheus.md | grep ‘android’

```
    public static final class BLECHAR {
        private static final UUID deviceOTA;
        private static final UUID deviceCommands;

        static {
            UUID fromString =
UUID.fromString("1163b0a20e-829b89b8aa98111111");
            deviceCommands = fromString;
            UUID fromString =
UUID.fromString("1163b0a20e-829b89b8aa98222222");
            deviceOTA = fromString;
        }
```

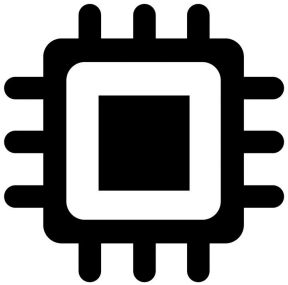
cat prometheus.md | grep ‘android’

```
public final class BLECommands {

    public void OTA (String url, String sha) {
        Intrinsics.checkNotNullParameter(url, "url");
        Intrinsics.checkNotNullParameter(sha, "sha");
        message = buildMessage(url, sha);
        send(message, BLECHAR.deviceOTA)
    }

    public String buildMessage() {
        return "{\"OTA\":{\"url\":\"" + this.url + "\",\"sha\":\"" + this.sha + "\"}}";
    }
```

cat prometheus.md | grep ‘android’

```
private String getFirmwareURL(){
  return "https://example.com/firmware/grill?version=1.0.1.bin"
}
```

IOT_Grill.bin

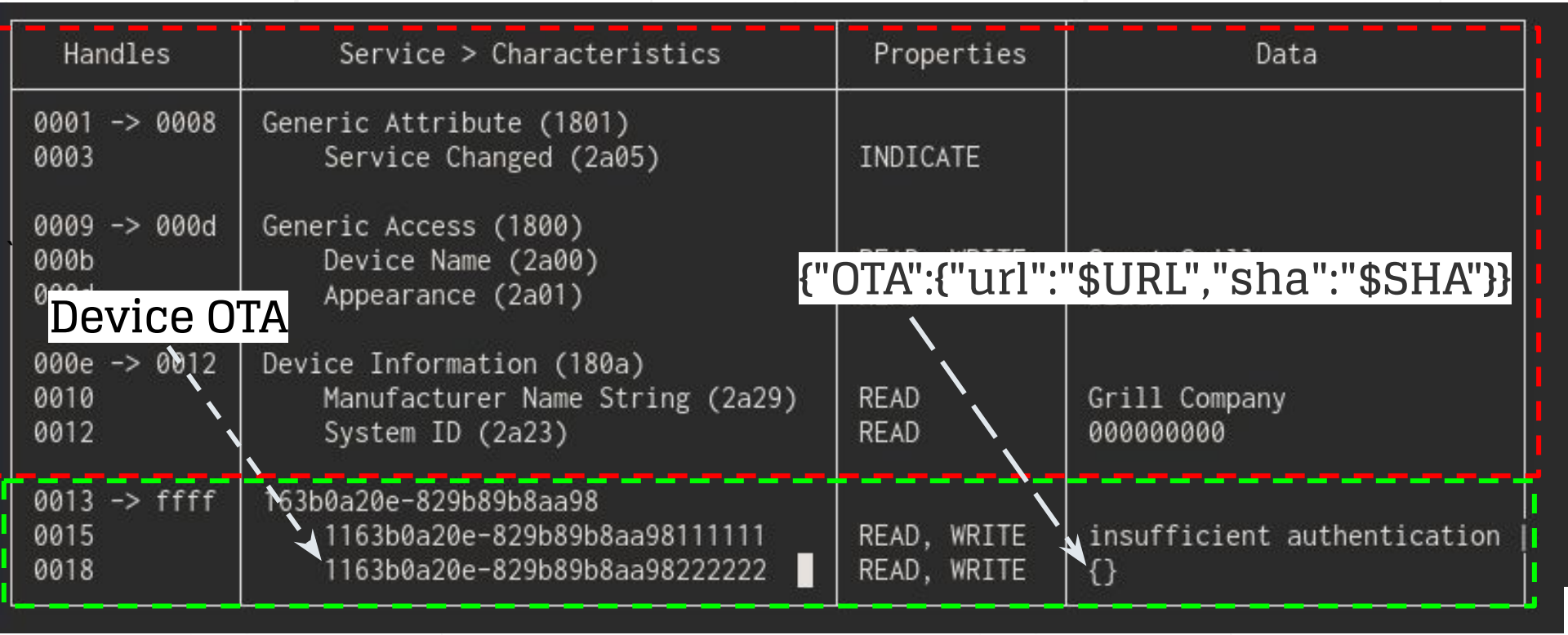


tldr prometheus.md

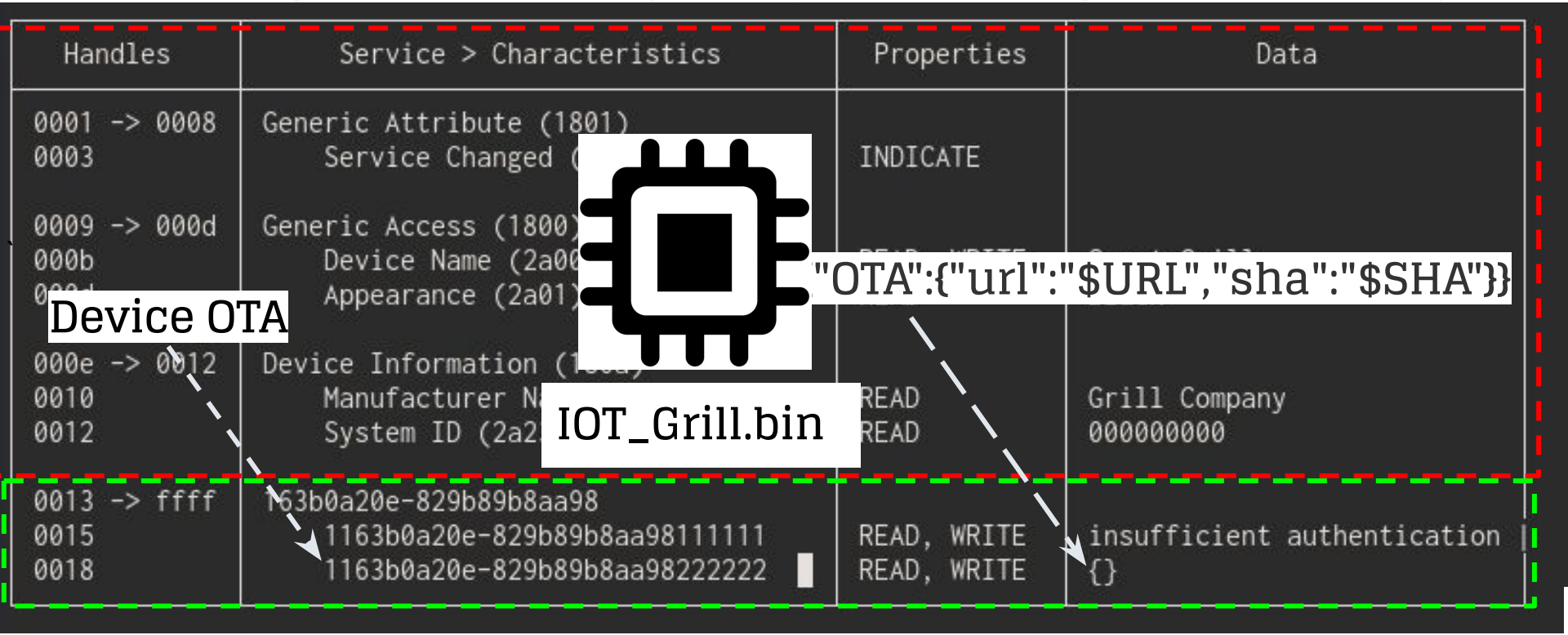
Handles	Service > Characteristics	Properties	Data
0001 -> 0008	Generic Attribute (1801)		
0003	Service Changed (2a05)	INDICATE	
0009 -> 000d	Generic Access (1800)		
000b	Device Name (2a00)	READ, WRITE	Smart Grill
000d	Appearance (2a01)	READ	Black
000e -> 0012	Device Information (180a)		
0010	Manufacturer Name String (2a29)	READ	Grill Company
0012	System ID (2a23)	READ	000000000
0013 -> ffff	1163b0a20e-829b89b8aa98		
0015	1163b0a20e-829b89b8aa98111111	READ, WRITE	insufficient authentication
0018	1163b0a20e-829b89b8aa98222222	READ, WRITE	{}

Device OTA

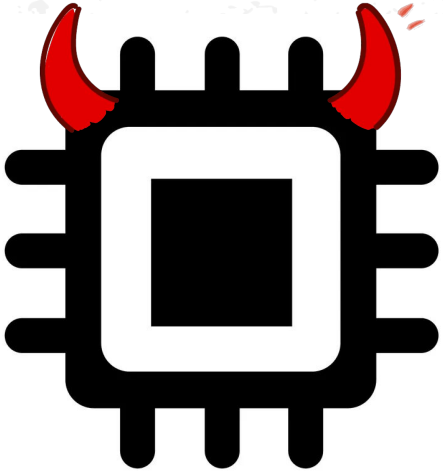
tldr prometheus.md



tldr prometheus.md



cat prometheus.md | grep ‘steps’



Create a malicious firmware image

Upload it via Bluetooth

Hack it

cat prometheus.md | grep ‘hacks?’

Hack it

- Turns out grills have serious safety features! Like preventing it from getting too hot... that’s implemented in software
- Safety features prevent the grill from being turned on over WiFi... would be nice to change that
- I like to customize things... maybe I could customize this?

cat prometheus.md | grep ‘hacks?’

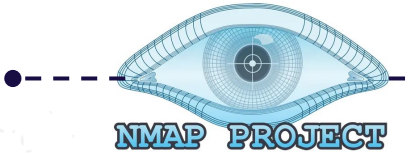
Hack it

→ Turns out grills have serious safety
ting it from
t’s implemented

ent the grill from
WiFi… would be

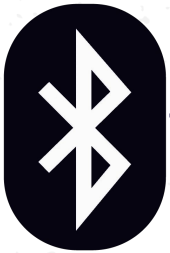
ngs… maybe I

tldr



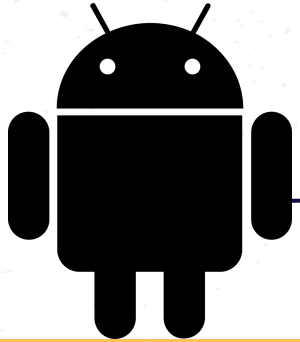
Web/Network

Start with the web/network interfaces



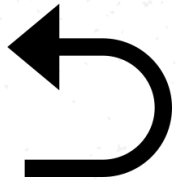
Bluetooth

See what you can do from the bluetooth interface



Mobile Apps

Get some more information



Start Over

Start over, but now with extra context and information to help you hack

wall ‘questions?’

bsides.carve.systems (ctf)

carvesystems.com/careers

@almostjson sotoventura.com