


# The smart home I didn't ask for

---

Nils Amiet

July 23, 2022

# Who am I?

- Nils Amiet
- Security researcher
- Privacy
- Data processing at scale
- Linux enthusiast
- @tmlxs 



# What is this about?

- This is my story with a smart home



## Chapter 0: Once upon a time...



## Once upon a time...

- I was looking for a new apartment
- Found an apartment for rent
- Decided to rent it
- The day I moved in, this was on the wall by the entrance

# The tablet



## Using the tablet

- Wall-mounted tablet
- Runs in kiosk mode, no apparent way to escape the app that it's running
- Touch buttons to control things in the apartment
  - Control window blinds
  - Control heating
  - Turn on/off lights
  - Open building entrance door when someone rings the doorbell

## Smart device dependence

- More and more buildings come pre-installed with smart devices
- Deep integration with the house/apartment
- Tenants become forced to use the smart device to do essential day-to-day tasks
  - For example, if outside window blinds are down, access to the balcony is blocked
  - The only way to raise the blinds is to use the smart device
- This creates a dependence on the smart device

## Smartphone pairing

- Android/iOS smartphone can be paired with the tablet
- Enable pairing of a new device on the tablet
  - Tablet displays 4-digit code valid for 60 seconds
- Enter code on smartphone to pair
- Once paired, the smartphone app can be used for remote control
- This works from anywhere on the internet

# Chapter 1: Network traffic analysis



- Android smartphone app named **eSMART Live**
- Produces encrypted traffic
- => Man-in-the-middle attack
- Used Pixel 4 smartphone, Android 11
- Rooted it with [Magisk](#)
- Since Android 7.0, apps only trust system certificates by default
  - Installed custom system certificates with the [Move Certificates](#) Magisk extension
  - Install Magisk modules directly in-app (Modules tab)

## Having a look at network traffic

- Produces 99% XMPP traffic, but also some HTTPS traffic
- HTTP traffic: [mitmproxy](#)
  - Also works with Burp suite or your favorite HTTPS proxy
- XMPP traffic: STARTTLS -> encrypted traffic
- Regular HTTPS proxies such as Burp or mitmproxy only support doing man-in-the-middle over HTTPS, not over XMPP
  - We needed another tool here
- XMPP proxy: [xmpp-mitm](#) worked for me



File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-F>

No.	Time	Source	Destination	Protocol	Length	Info
54	4.749689866	54.76.204.176	192.168.12.36	TCP	66	5222 → 42280 [ACK] Seq=4202 Ack=2087 Win=64128 Len=0 TSv
55	4.880823832	54.76.204.176	192.168.12.36	XMPP/XML	203	PRESENCE < a-280074a2fe917-686@myesmart.net/Smack
56	4.983432417	54.76.204.176	192.168.12.36	XMPP/XML	1514	IQ(result) QUERY(jabber:iq:roster) < a-280074a2fe917-686@
57	5.013552794	192.168.12.36	54.76.204.176	TCP	66	42280 → 5222 [ACK] Seq=2087 Ack=4339 Win=118528 Len=0 TS
58	5.013553191	192.168.12.36	54.76.204.176	TCP	66	42280 → 5222 [ACK] Seq=2087 Ack=5787 Win=121344 Len=0 TS
59	5.013648761	54.76.204.176	192.168.12.36	XMPP/XML	371	MESSAGE < 00-00-15-05-05-01-58@conference.myesmart.net
60	5.014653605	192.168.12.36	54.76.204.176	TCP	66	42280 → 5222 [ACK] Seq=2087 Ack=6092 Win=124416 Len=0 TS
61	5.022103413	192.168.12.36	54.76.204.176	XMPP/XML	285	PRESENCE > 00-00-15-05-05-01-58@conference.myesmart.net/

Transport Layer Security

- ↳ TLSv1.3 Record Layer: Application Data Protocol: xmpp
- ↳ XMPP Protocol
  - ↳ MESSAGE [type="normal"]
    - from: 00-00-15-05-05-01-58@conference.myesmart.net
    - to: a-280074a2fe917-686@myesmart.net/Smack
    - type: normal
    - ↳ BODY [value="00-00-15-05-05-01-58@myesmart.net/Smack invites you to the room 00-00-15-05-05-01-58@conference.myesmart.net"]
      - value: 00-00-15-05-05-01-58@myesmart.net/Smack invites you to the room 00-00-15-05-05-01-58@conference.myesmart.net (00-00-15-05-05-01-58@myesmart.net/Smack invites you to the room 00-00-15-05-05-01-58@conference.myesmart.net)
    - ↳ X MUC-USER []
    - ↳ X [jid="00-00-15-05-05-01-58@conference.myesmart.net" xmlns="jabber:x:conference"] [UNKNOWN]
      - ↳ jid: 00-00-15-05-05-01-58@conference.myesmart.net [UNKNOWN ATTR]
        - ↳ [Expert Info (Note/Undecoded): Unknown attribute jid]

```

0000 3c 6d 65 73 73 61 67 65 20 66 72 6f 6d 3d 27 30 <message from='0
0010 30 2d 30 30 2d 31 35 2d 30 35 2d 30 31 00-15- 05-05-01
0020 2d 35 38 40 63 6f 6e 66 65 72 65 6e 63 65 2e 6d -58@conf erence.m
0030 79 65 73 6d 61 72 74 2e 6e 65 74 27 20 74 6f 3d yesmart. net' to=
0040 27 61 2d 32 38 30 30 37 34 61 32 66 65 39 31 37 'a-28007 4a2fe917
0050 2d 36 38 36 40 6d 79 65 73 6d 61 72 74 2e 6e 65 -686@mye smart.ne
0060 74 2f 53 6d 61 63 6b 27 20 74 79 70 65 3d 27 6e t/Smack' type='n
0070 6f 72 6d 61 6c 27 3e 3c 78 20 78 6d 6c 6e 73 3d ormal'>< x xmlns=
0080 27 68 74 74 70 3a 2f 2f 6a 61 62 62 65 72 2e 6f 'http:// jabber.o
  
```

Frame (371 bytes) Reassembled TCP (554 bytes) Decrypted TLS (532 bytes)

autoconnect-light1on-light2-on-blind1-orientation-a-b-light-off-lightb-off.pcapng

Packets: 320 · Displayed: 320 (100.0%) Profile: Default

## Software Wi-Fi access point

- Software Wi-Fi AP on laptop: [linux-wifi-hotspot](#)
  - Connect to that Wi-Fi AP using the rooted smartphone
- It will create a new network interface named ap0

- Setup mitmproxy to obtain clear-text HTTPS traffic
- The SSLKEYLOGFILE can then be used with Wireshark to decrypt traffic
  - Edit > Preferences > Protocols > TLS > (Pre)-Master-Secret log filename > Browse... and select sslkeylogfile.txt path
- Listens on port 8080 by default

## Running mitmproxy

```
export SSLKEYLOGFILE=~/.esmart/mitm/sslkeylogfile.txt  
mitmweb --mode transparent --showhost
```

- Generate a new key pair and certificate
  - The built-in ones did not work for me
  - Apparently, app checks for the certificate's domain name
  - Had to create a certificate valid for `xmpp.myesmart.net`

## Generating a new certificate

```
openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 \  
-keyout private.key -outform pem -out server.pem \  
-subj "/CN=xmpp.myesmart.net"  
openssl x509 -in server.pem -out server.crt -outform der
```

## XMPP proxy (part 2)

- Install `server.crt` on the Android device as user certificate
- Use Magisk extension to move it to system certificate
- Run `xmpp-mitm`
- Also produces an `sslkeylogfile.txt`

### Running `xmpp-mitm`

```
sudo ./xmpp_mitm.py --iface ap0 --write_file out.pcap \  
  --sslkeylog ~/esmart/mitm/sslkeylogfile.txt --port 5222 \  
  --key private.key --cert server.pem
```

# Transparent proxy setup

- Transparent proxy is set by redirecting traffic automatically

## Transparent proxy using iptables

```
$ export IFACE=ap0
$ iptables -t nat -A PREROUTING -i $IFACE -p tcp \
  --dport 80 -j REDIRECT --to-port 8080
$ iptables -t nat -A PREROUTING -i $IFACE -p tcp \
  --dport 443 -j REDIRECT --to-port 8080
$ iptables -t nat -A PREROUTING -i $IFACE -p tcp \
  --dport 5222 -j DNAT --to-destination 192.168.12.1
```

- No certificate pinning
  - We were able to see the clear-text HTTPS and XMPP traffic
  - JSON payloads are sent inside XMPP messages

## Example payload (turn on light 13)

```
{ "headers": {  
  "from": "a-280074a2fe917-686",  
  "to": "master",  
  "timestamp": "2021-11-04 16:09:18Z",  
  "method": "CMD",  
  "type": "operation",  
  "size": 22,  
  "version": "1.13.0"  
},  
  "body": {  
    "id": 13,  
    "onoff": "on"  
  }  
}
```





- App source code is obfuscated
  - This gives us limited understanding of that app
  - Can reverse engineer but there may be quicker ways around this
  - I decided to move on

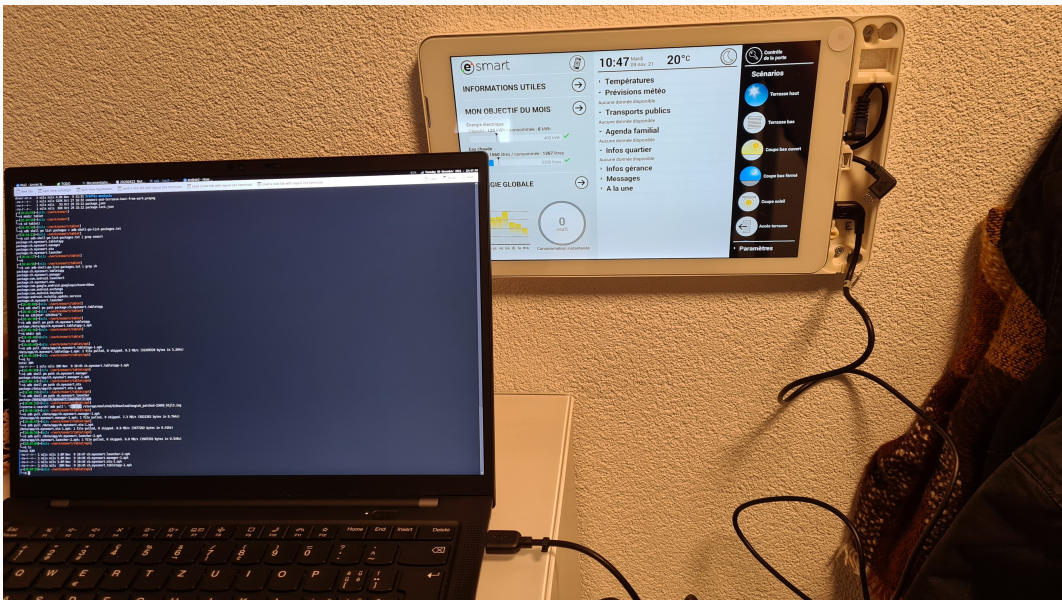
## Chapter 2: Tablet analysis



- Goal: figure out how the tablet works
- Can we connect to the tablet?















- USB debugging is enabled by default
- Tablet is rooted by default (!)
- Runs Android 4.4 (released October 2013)

## Collecting .apk files for installed apps

### Enumerate installed apps

```
$ adb shell pm list packages
package:ch.myesmart.tabletapp
...
package:com.teslacoilsw.quicksshd
```

### Get path of installed apk

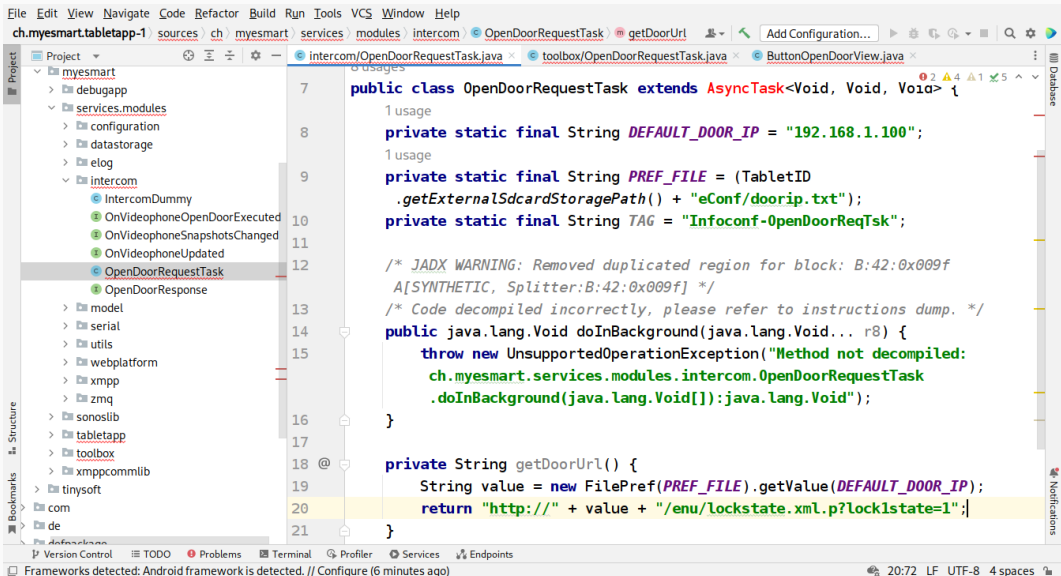
```
$ adb shell pm path ch.myesmart.tabletapp
package:/data/app/ch.myesmart.tabletapp-1.apk
```

### Get an .apk from its package name

```
$ adb pull /data/app/ch.myesmart.tabletapp-1.apk
```

# Unpacking and decompiling .apk files

- Target:
  - 4 eSMART apps: tabletapp, manager, ota, launcher
  - SSH server app: quicksshd
- Unpack/Decompile APK: [jadx](#)
  - Was good enough for me
  - Alternative
    - APK -> JAR: [enjarify](#)
    - Use classical java decompiler, for example: [fernflower](#) (from IntelliJ)
- eSMART apps code was non-obfuscated
  - Easy to understand what happens



## Different tablet versions

- The tablet app source code indicates that there are multiple versions of tablets that are deployed
- Mine was one of the oldest (rk3188) but there are also others
  - Rockchip rk3188, C91, C93, Finch

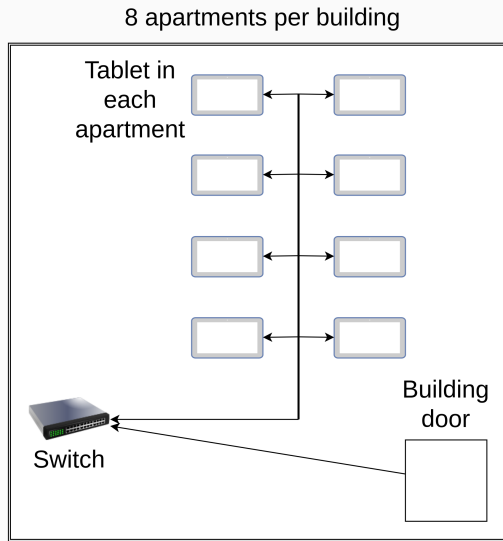
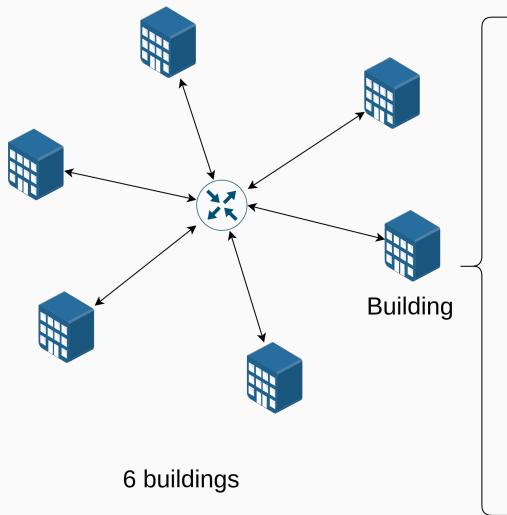
## Building's main entrance door

- Pulled files on the tablet in /data/data, /mnt/external\_sd, /mnt/internal\_sd
- File /data/data/ch.myesmart.tabletapp/files/eConf/doorip.txt contains the IP address of the building's main entrance door (10.0.5.100)
- Open door: simple HTTP GET request
  - No authentication required
  - Only have to be connected to the same wired network

### Opening my building's main entrance door

```
$ curl http://10.0.5.100/enu/lockstate.xml.p?lock1state=1
```

# Neighborhood



## Other doors in the neighborhood

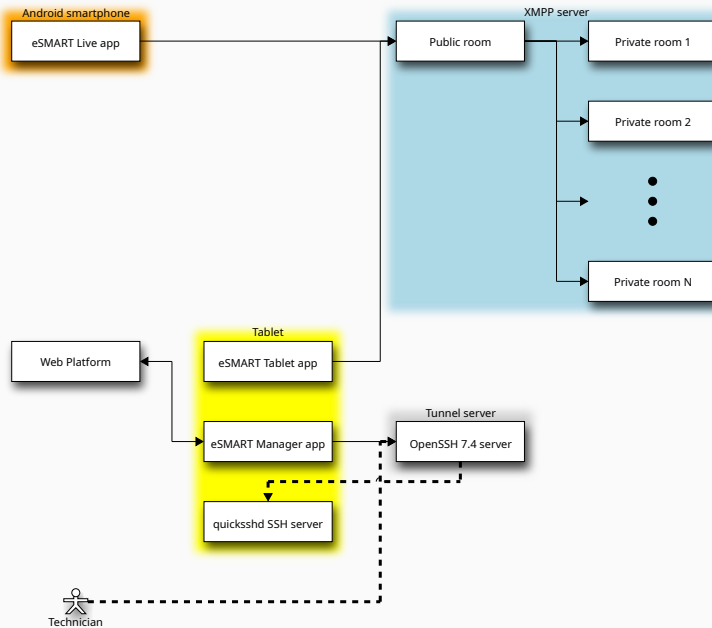
- Shared wired local IP network for all apartments in neighborhood
- Ping from my tablet to other tablets/doors works
- Can open own building's door and ping other doors
  - Suggests other 5 building doors can be opened
  - Did not proceed to exploitation for legal reasons
  - Vendor did not deny
- Tablets/doors IP is logical
  - Tablet B5 has IP 10.0.2.5
  - Building B door is 10.0.2.100
- Note: apartment doors require a physical key to be opened



## Chapter 3: Findings



# Overall system architecture



- Smartphone app connects to XMPP server at `xmpp.myesmart.net`
- Unique Jabber ID (JID) derived from random value + MAC address
- Uses password-based authentication (SASL with DIGEST-MD5 mechanism)
  - Username = JID
  - Password = SHA512(JID + Random value)
- XMPP server registration is open
  - Newly deployed tablets generate credentials and register on first run
- Both tablet and smartphone join a public room on the XMPP server
- Tablet also joins a private room named as `{JID}@conference.myesmart.net`
- When pairing is successful, the tablet invites the smartphone to its private room
- The tablet interprets all messages received in that private room

## Possibility for remote exploitation?

- Private room name is basically JID + hardcoded string
  - The public room may be used to enumerate tablets and find room names
- If we know a tablet's JID, can we join the private room?
  - I have reported this potential issue to the vendor
  - Another researcher had already reported it, and they patched it early 2021
  - This was an actual issue in the past!
  - Rooms are now invite-only

## Web platform

- Tablet app listens for commands sent through a web app at `webplatform.myesmart.net`



Please sign in

Sign in

## SSH tunnel server

- One webplatform command tells tablet to SSH remote port forward on `tunnel.myesmart.net`
- eSMART technician can use opened port to connect back into tablet
- SSH server uses password authentication, whose MD5 hash is stored in a text file on the tablet. Was unable to brute force it (too long)
- However, SSH server runs on the tablet, meaning we control the server
  - Modify SSH server so that it logs passwords
  - Call eSMART hotline, pretend there is an issue with my tablet
  - Wait for technician to remotely troubleshoot, collect password
- All tablets in same deployment/neighborhood have the same password
  - Confirmed 2 tablets have same hash, vendor also confirmed same hash for all neighborhood
- Tablets have a microphone and camera. . . privacy issue, and are rooted => can control apartment

- Use public key authentication instead of passwords
  - This way, even if the server is untrusted, the private key cannot be compromised
- Turn off SSH server on tablet
  - It doesn't need to run at all times, only start it when necessary

- The building owner pays for internet access used by tablets
  - Enable Wi-Fi hotspot on tablet => get free internet access
- Pairing PIN code spamming
  - Spam all possible PIN codes (only 4-digits)
  - Wait for someone to start pairing, send PINs fast, get paired first



## Chapter 4: Disclosure



## Findings summary

- USB debugging enabled by default
- Tablet is rooted
- Can open door of other buildings if on same network
- Usage of Android 4.4 (no more security patches)
- Tablet apps are not obfuscated
- No certificate pinning on Android smartphone app
- SSH server password authentication
- No pairing PIN code rate limiting

## Disclosure timeline

- December 6, 2021: Disclosure to eSMART team via email
- December 14, 2021: Acknowledge receipt
- December 15, 2021: Receive response to disclosed items
- December 16, 2021: Offer to discuss/clarify findings/impact in person after the holiday
- January 25, 2022: Send reminder to meet in person
- January 27, 2022: Settle on meeting date and time
- February 15, 2022: Meet in person
- May 10, 2022: Notify eSMART team that this talk was accepted
- May 11, 2022: Receive response saying fix is in progress and will be deployed this month if final tests are successful
- July 2022: USB debugging disabled, quicksshd uninstalled from tablet

# Conclusions

- Smart devices should be built with security in mind from the start
- Deep integration with house/apartment leads to even worse consequences in case of breach or failure
- If you want cool research, put smart hardware in a security researcher's house

Thank you

Questions?