LOCKCON

Bluetooth Locks
(and an update on the X-09)

by mh, in cooperation with Ray

2019-10-25
LockCon 2019, Baarlo, NL

[V1.1 as of 2019-11-05 with some updates that were added after LockCon 2019.]

This is a modified version of a talk at CCCamp2019 and Black Hat USA 2019 Briefings.

Track

Speakers

Popular

Most Viewed

Highest Rated

Moving from
Europe's Hi
into One of
-117)

Ray

Michael Hue

2019 US

Track: Hardware

Show more

What This Presentation Is About

- “Smart” devices using Bluetooth Low Energy
- How to analyze / hack / improve them
- Vulnerabilities we found that way, from cheap padlocks to hotel door systems
- An update on the X-09 high security lock

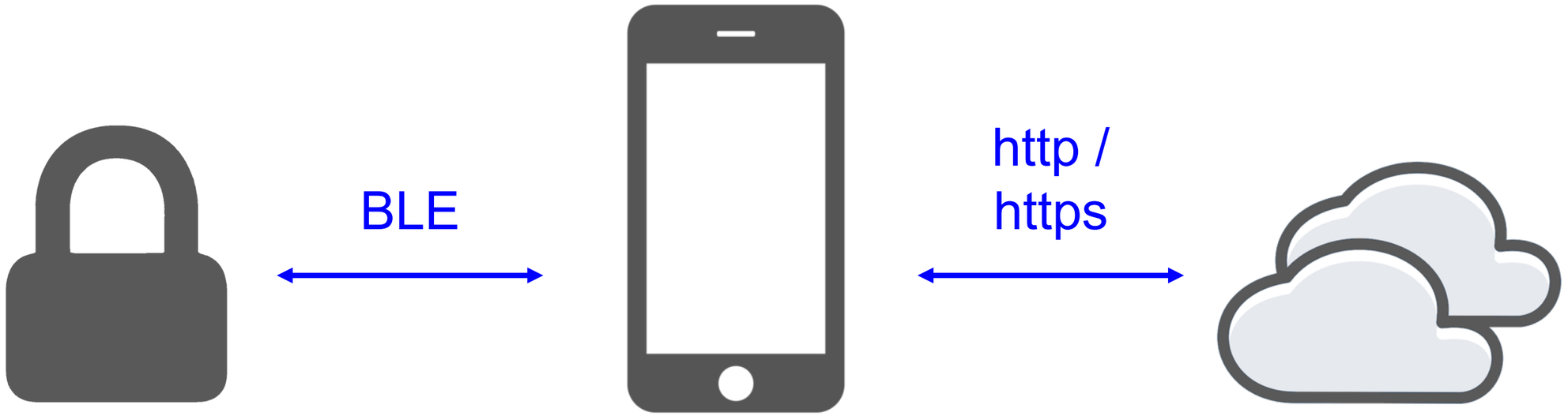
Agenda

1. Bluetooth Low Energy (BLE) Ecosystem
2. How to Analyze BLE Systems
3. Previous Vulnerabilities
4. BLE Hotel Keys
5. Responsible Disclosure
6. X-09 Side-channel Attack?

The BLE Ecosystem

BLE Locks

Components of a “Smart” Lock Ecosystem:

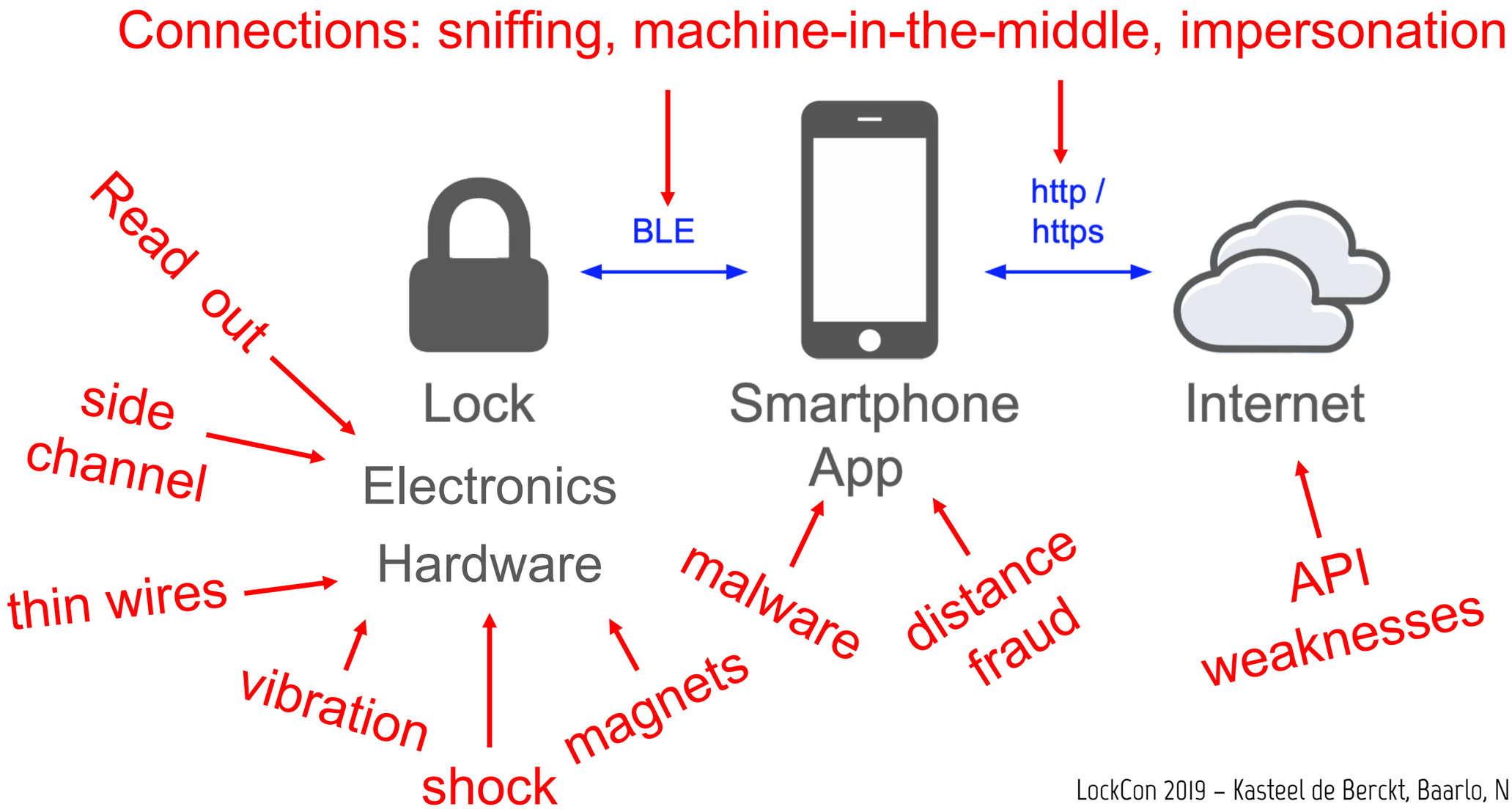


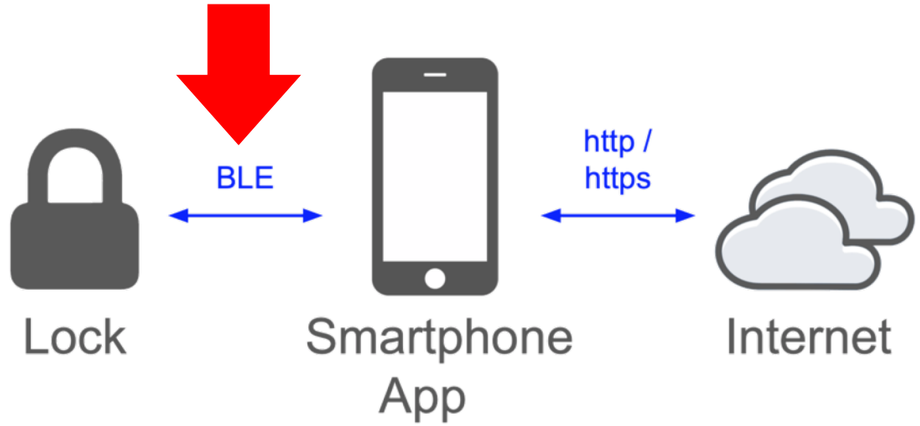
Lock

Smartphone App

Internet

BLE Locks - Attack Vectors





How to Analyze BLE

Getting the BLE Traffic

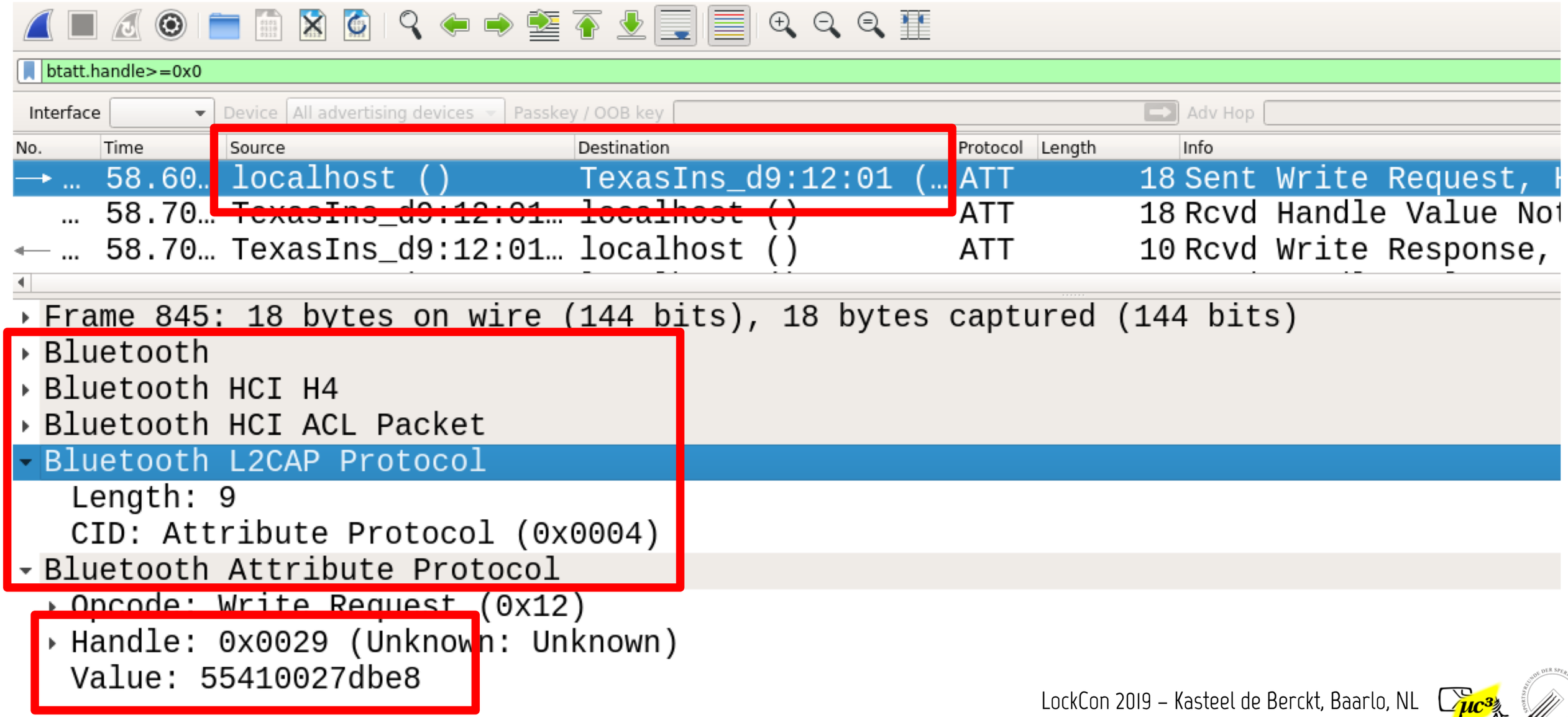
- On your own device, log traffic locally:
  - Android: enable debug mode, activate HCI snoop log
  - iOS: install Apple Bluetooth Debug Certificate on your device

Getting the BLE Traffic

- Now use the app and interact with the device
- Note timestamps of important actions (like “open lock”)
- Get HCI log from phone
- Analyze using tools like Wireshark

LOCKCON

BLE in Wireshark

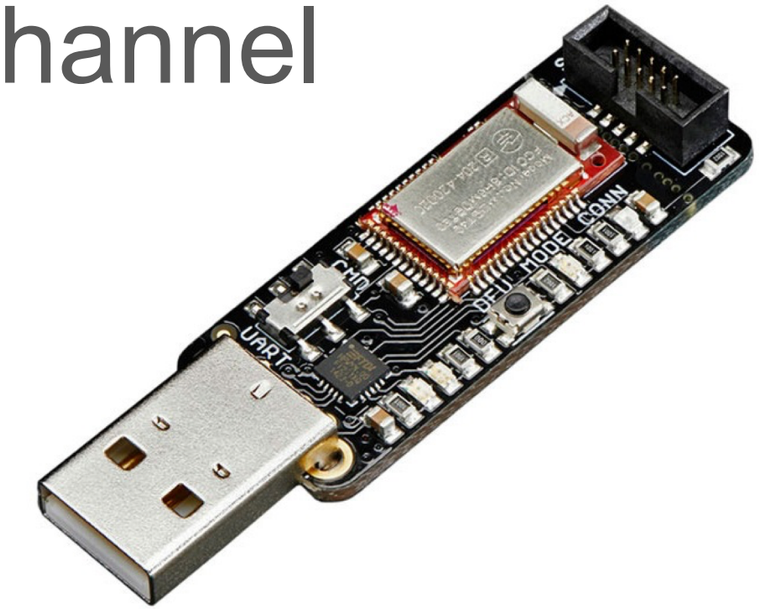


Sniffing BLE

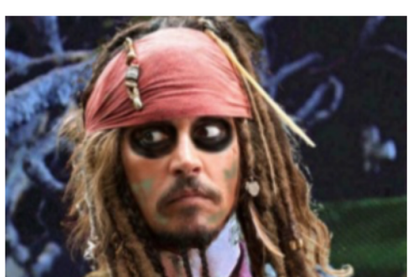
- For real attacks, sniff BLE over the air
- 3 advertising channels, need to listen to the active one to catch a connection setup
- USB BLE sniffers ~$25

Classic Sniffing Tools

- Adafruit Bluefruit LE Sniffer or Ubertooth One
- Support Wireshark live view
- Can monitor only 1 advertising channel at a time, follow sequence
- OK for proof of concept, for reliable attacks you need more



Our Favorite Tool: btlejack



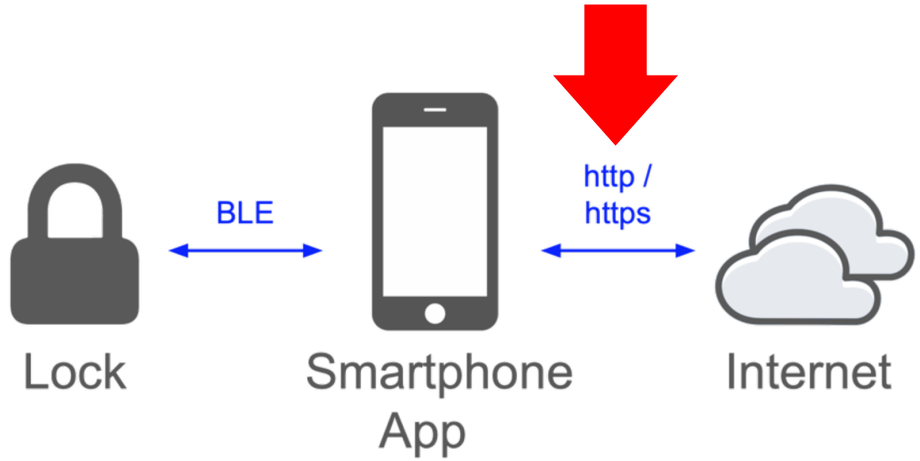
- btlejack by Damien Cauquil
- Firmware for cheap BLE USB devices: BBC Micro:Bit, BLE400, Adafruit Sniffer
- Use 3 devices and follow all advertising channels in parallel
- Much more than just sniffing: hijacking, …

Ray's Proof-of-Concept

mh's Slightly Optimized Setup

New Tool: Mirage

- Mirage by Romain Cayre
- brings its own (hackable) BLE stack
  → more transparent MITM
- MITM on one device only (good & bad)
- Powerful and flexible framework
  → more difficult to use



How to Analyze the Backend Link

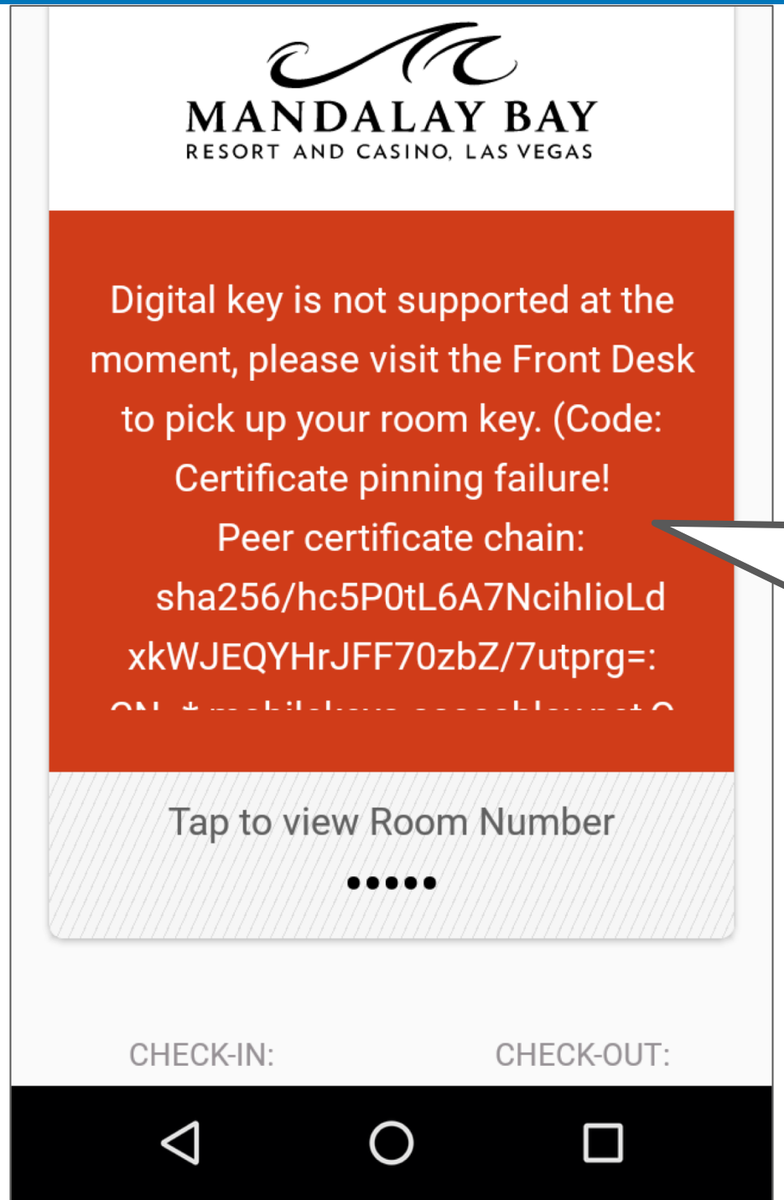
TLS MITM

- Only few apps use plain HTTP
- Add fake root CA to intercept TLS/HTTPS
- MITM tools create certificates on the fly
- To analyze app, not to break other people's TLS

Using MITM CAs

- iOS: just declare it as trusted
- Android:
  - works easily up to 6.x,
    needs rooted device on >=7
  - or modify app to use user cert store:
    add `network_security_config` to manifest (then rebuild, sign)

If the App Uses Certificate Pinning



moment, please visit the Front Desk to pick up your room key. (Code: Certificate pinning failure!

If the App Uses Certificate Pinning

- Try the other app (iOS vs. Android), or an older version Android app
- Modify the app, rebuild, sign
- Use Frida / objection
  - intercept calls in the app, or in the OS
    → unlimited possibilities :)

FRIDA

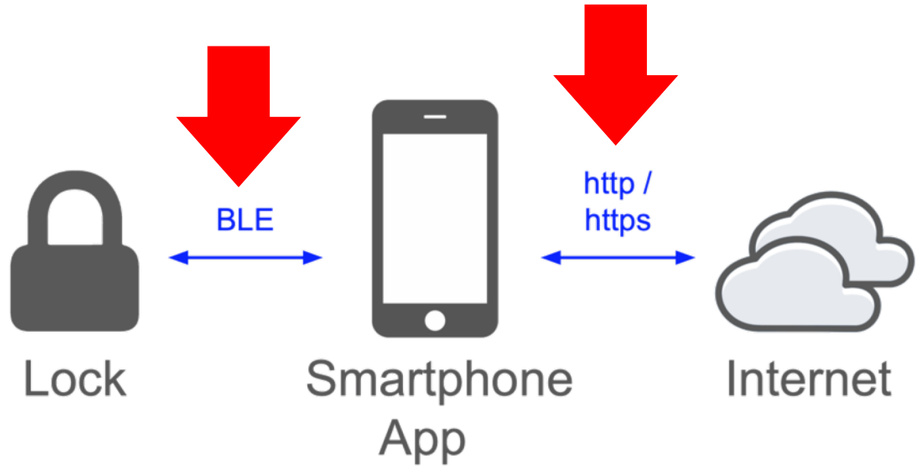
OBJECTION
RUNTIME MOBILE EXPLORATION
GIT.IO/OBJECTION

TLS MITM Tools

- Unix command line: mitmproxy
- macOS: Charles Proxy
- Many more available, like Burp Suite or Fiddler

Example: mitmproxy

```
2016-12-26 04:33:20 POST https://nokeapp.com/
                         ← 200 OK text/html 940b 762ms
          Request                Response                    Detail
Content-Type:              text/html; charset=utf-8
X-Cloud-Trace-Context:     c6d3795272d60331a34ca3e03922c271
Date:                      Mon, 26 Dec 2016 04:57:55 GMT
Server:                    Google Frontend
Content-Length:            940
Connection:                close
JSON                                                    [■:JSON]
{
    "lockcount": 2,
    "locks": [
        {
            "autounlock": "0",
            "battery": "196",
            "fobcodesavailable": "25",
            "fobcodesrefreshstate": "",
            "foblocklinks": [],
            "foblocklinkscount": "0",
            "lockid": "38850",
            "lockkey": "40637020F41C",
[6/71]                                          ?:help q:back [*:21984]
```



Analyzing the Collected Data

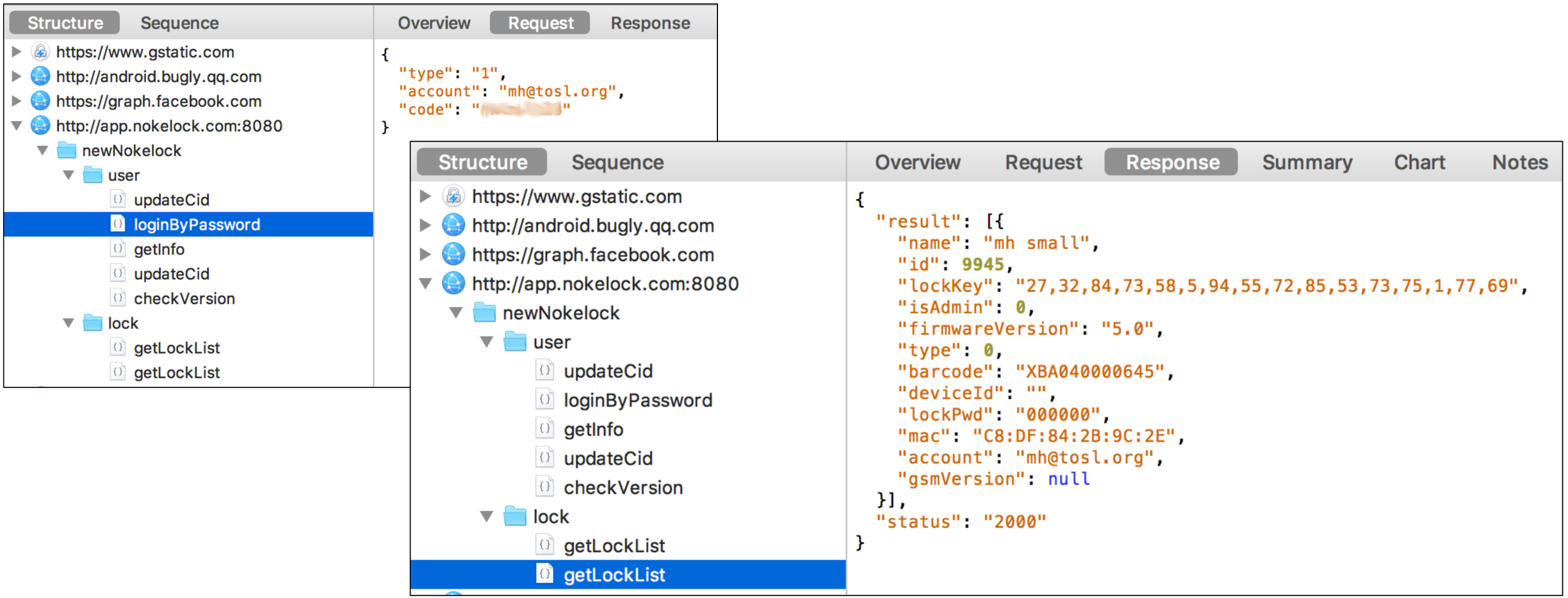
Example: Nokelock

- Small, cheap BLE padlock
- Company offers a large variety of locks (also for doors, cabinets, bikes, e-scooters…)

Note: Research as of 2018, the app has been improved in the meantime.

Analyzing the Collected Data - HTTPS

Unencrypted HTTP traffic:



16 bytes “lockKey”

Structure | Sequence

- https://www.gstatic.com
- http://android.bugly.qq.com
- https://graph.facebook.com
- http://app.nokelock.com:8080
  - newNokelock
    - user
      - updateCid
      - loginByPassword
      - getInfo
      - updateCid
      - checkVersion
    - lock
      - getLockList
      - getLockList

Overview | Request | Response | Summary | Chart | Notes


```
{
  "result": [{
    "name": "mh small",
    "id": 9945,
    "lockKey": "27,32,84,73,58,5,94,55,72,85,53,73,75,1,77,69",
    "isAdmin": 0,
    "firmwareVersion": "5.0",
    "type": 0,
    "barcode": "XBA040000645",
    "deviceId": "",
    "lockPwd": "000000",
    "mac": "C8:DF:84:2B:9C:2E",
    "account": "mh@tosl.org",
    "gsmVersion": null
  }],
  "status": "2000"
}
```


16 bytes “lockKey”

`1B 20 54 49 3A 05 5E 37`
`48 55 35 49 4B 01 4D 45`

→ maybe AES-128?

Traffic Looked Random → Decrypt It

Decrypt BLE traffic with AES-128 ECB

→ doesn't look random → ✓

```
06 01 01 01 5d 1a 79 5c 5c 51 77 13 10 79 04 74   (app → lock)
06 02 07 d4 9c ea ce 01 05 00 00 00 00 00 00 00   (lock → app)
02 01 01 01 d4 9c ea ce 7c 3f 2b 34 4b 11 5b 4d   (app → lock)
02 02 01 59 9c ea ce 01 05 00 00 00 00 00 00 00   (lock → app)

05 01 06 30 30 30 30 30 30 d4 9c ea ce 1f 7e 10   (app → lock)
05 02 01 00 9c ea ce 01 05 00 00 00 00 00 00 00   (lock → app)
05 0d 01 00 9c ea ce 01 05 00 00 00 00 00 00 00   (lock → app)

05 01 06 30 30 30 30 30 30 d4 9c ea ce 07 10 0a   (app → lock)
05 02 01 00 9c ea ce 01 05 00 00 00 00 00 00 00   (lock → app)
05 0d 01 00 9c ea ce 01 05 00 00 00 00 00 00 00   (lock → app)
```

Analyzing the Protocol

Look for patterns

(compare several sessions):

```
06 01 01 01 5d 1a 79 5c 5c 51 77 13 10 79 04 74   (app → lock)
06 02 07 d4 9c ea ce 01 05 00 00 00 00 00 00 00   (lock → app)
02 01 01 01 d4 9c ea ce 7c 3f 2b 34 4b 11 5b 4d   (app → lock)
02 02 01 59 9c ea ce 01 05 00 00 00 00 00 00 00   (lock → app)

05 01 06 30 30 30 30 30 30 d4 9c ea ce 1f 7e 10   (app → lock)
05 02 01 00 9c ea ce 01 05 00 00 00 00 00 00 00   (lock → app)
05 0d 01 00 9c ea ce 01 05 00 00 00 00 00 00 00   (lock → app)

05 01 06 30 30 30 30 30 30 d4 9c ea ce 07 10 0a   (app → lock)
05 02 01 00 9c ea ce 01 05 00 00 00 00 00 00 00   (lock → app)
05 0d 01 00 9c ea ce 01 05 00 00 00 00 00 00 00   (lock → app)
```

Analyzing the Protocol

Deduce protocol (from a few sessions):

AUTH_REQUEST	**(060101)**,	random padding	(app → lock)
AUTH_RESPONSE	**(060207)**,	4 byte session ID, 0 padding	(lock → app)
STATUS_REQUEST	**(020101)**,	4 byte session ID, random padding	(app → lock)
STATUS_RESPONSE	**(020201)**,	batt state, 3 byte sess.ID, 0 padding	(lock → app)
UNLOCK_REQUEST	**(050106)**,	*passcode*, session ID, random padding	(app → lock)
UNLOCK_ACK	**(050201)**,	3 byte session ID, 0 padding	(lock → app)
UNLOCK_CONFIRM	**(050d01)**,	3 byte session ID, 0 padding	(lock → app)

→ Session replay protection: 4 byte session ID created by the lock.

Next Steps

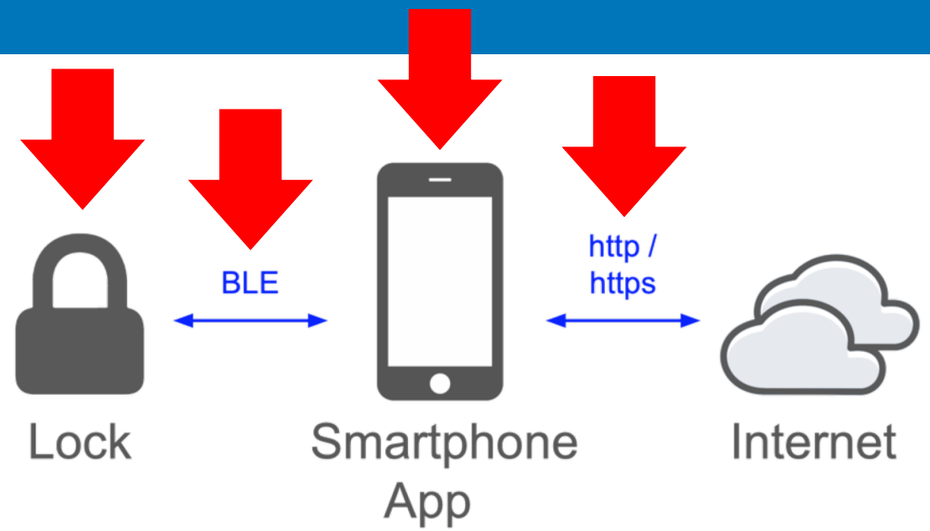
Verify the findings, look for weaknesses.

BLE protocol

- Write SW that mimics the app, e.g. Python, bluepy or Adafruit_BluefruitLE
- Explore the protocol, use fuzzing techniques

Whole system

- Maybe an OEM uses the same key for all devices?
- Maybe the backend leaks other users' keys?
  (when researching this, consider legal restrictions!)



Examples of Previous VULNs

ANBOUD Padlock

- Typical cheap BLE padlock
- Shim-proof mechanics, but passcode transmitted in plain text
- To our knowledge still unfixed

ANBOUD PWNED

```
▾ Bluetooth Attribute Protocol
  ▸ Opcode: Write Request (0x12)
  ▸ Handle: 0x0029 (Unknown: Unknown)
    Value: 55410027dbe8
```

- Hex 0x027db = Dec 010203
- That’s the code I set on the lock
- Original app can now be used to open lock with sniffed code

~~12~~ 14 of 16 locks vulnerable

- Rose & Ramsey at DefCon 24 (2016)
- 12 of 16 tested locks had simple BLE vulnerabilities
- Only two of the padlocks remained unbroken
- One of those we opened with a magnet, like its predecessor, ...


[In the presentation we had a video showing
how to turn the internal motor with a strong magnet.
This PDF does not include the video,
but you can get an idea from the video that’s linked on the previous slide.]
0:21,07

~~12~~ 14 of 16 locks vulnerable

- Rose & Ramsey at DefCon 24 (2016)
- 12 of 16 tested locks had simple BLE vulnerabilities
- Only two of the padlocks remained unbroken
- One of those we opened with a magnet, like its predecessor, the other one ...

NOKĒ Padlock (!= Nokelock)

- One of the first BLE padlocks, created on Kickstarter in 2014

- Note: Research applies to the original firmware from 2015-2017
  (Our responsible disclosure 2016 led to a firmware update in 2017)

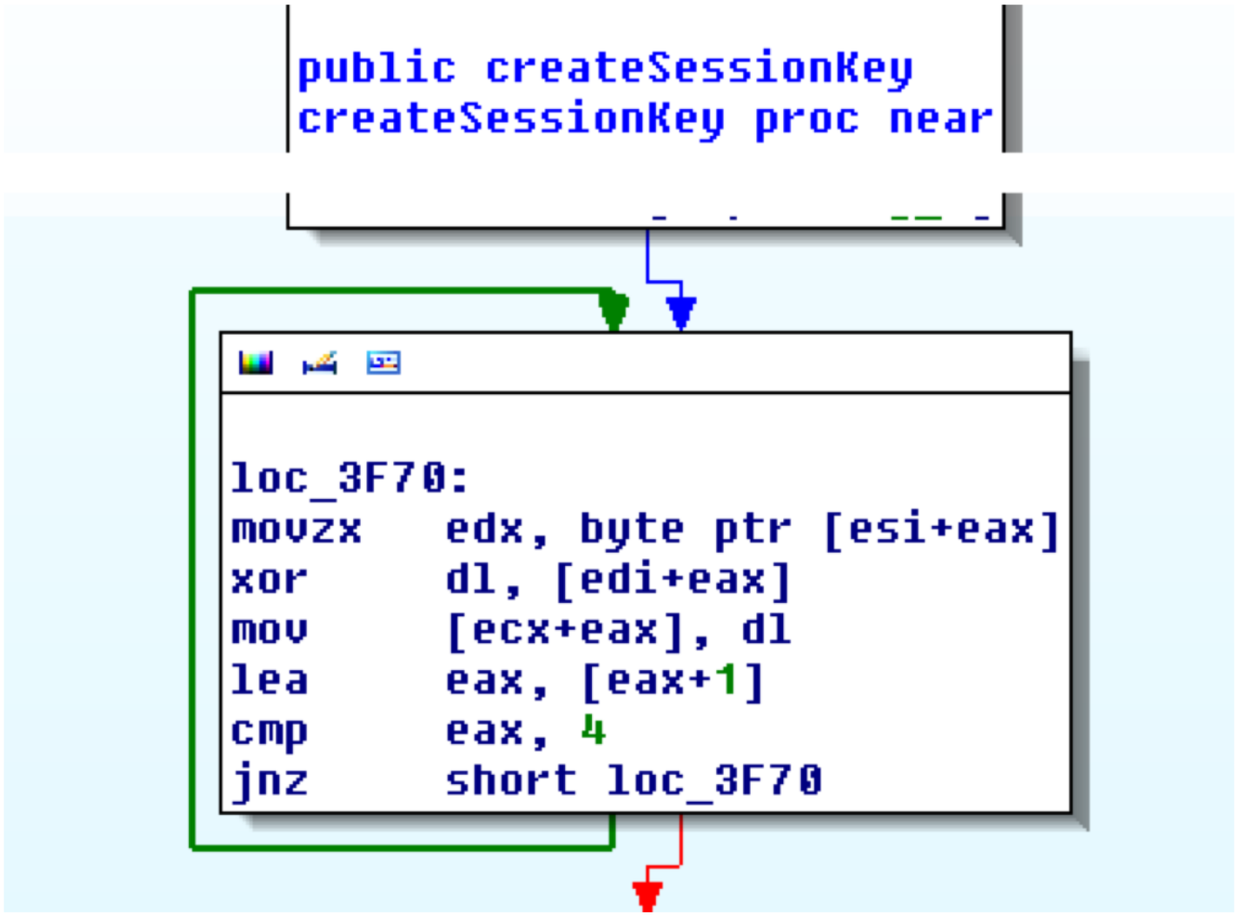
NO(KĒ) Security

- Uses AES-128 cipher
- Uses two different secrets for owner and other users
- Time restrictions only enforced in app

NOKĒ AES VULN

- Secret is transmitted using individual AES session keys
- But session keys are created in a “secret handshake” using a hardcoded AES key
- Security by obscurity

NOKĒ Session Key



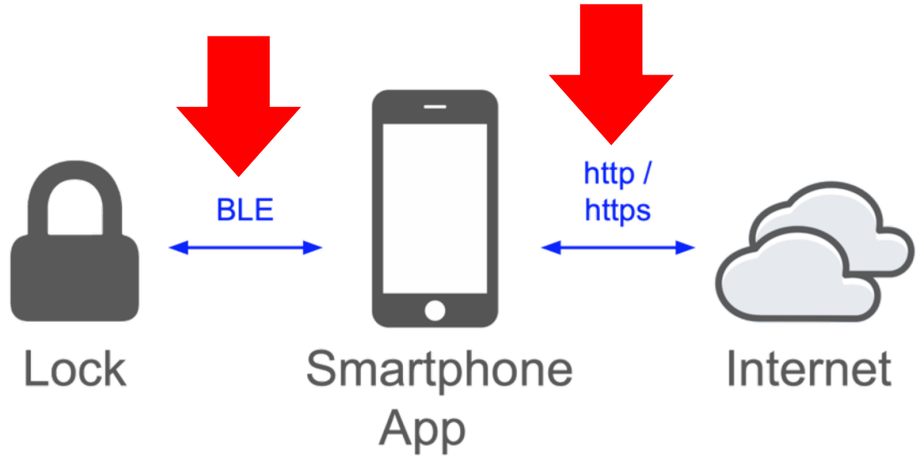
...from binary .so file in APK

NOKĒ KEX Broken

```
app nonce:    b14c68a1
                XOR
lock nonce:   bff91ae4
           =  0eb57245
                +      (add byte-by-byte modulo 256)
  0001020304 05060708  090a0b0c0d0e0f  (pre shared key)

=
  0001020304 13bb794d  090a0b0c0d0e0f  (new session key)
```

New session key can now be used to decrypt transfer of the user's secret



BLE Hotel Keys

Why BLE for Hotels?

- Main purpose: self-check-in
- No keycard anymore, mobile phone app is the key
- Hotels can reduce front desk staff
- Guests don’t have to wait in queue

Challenges for Vendors

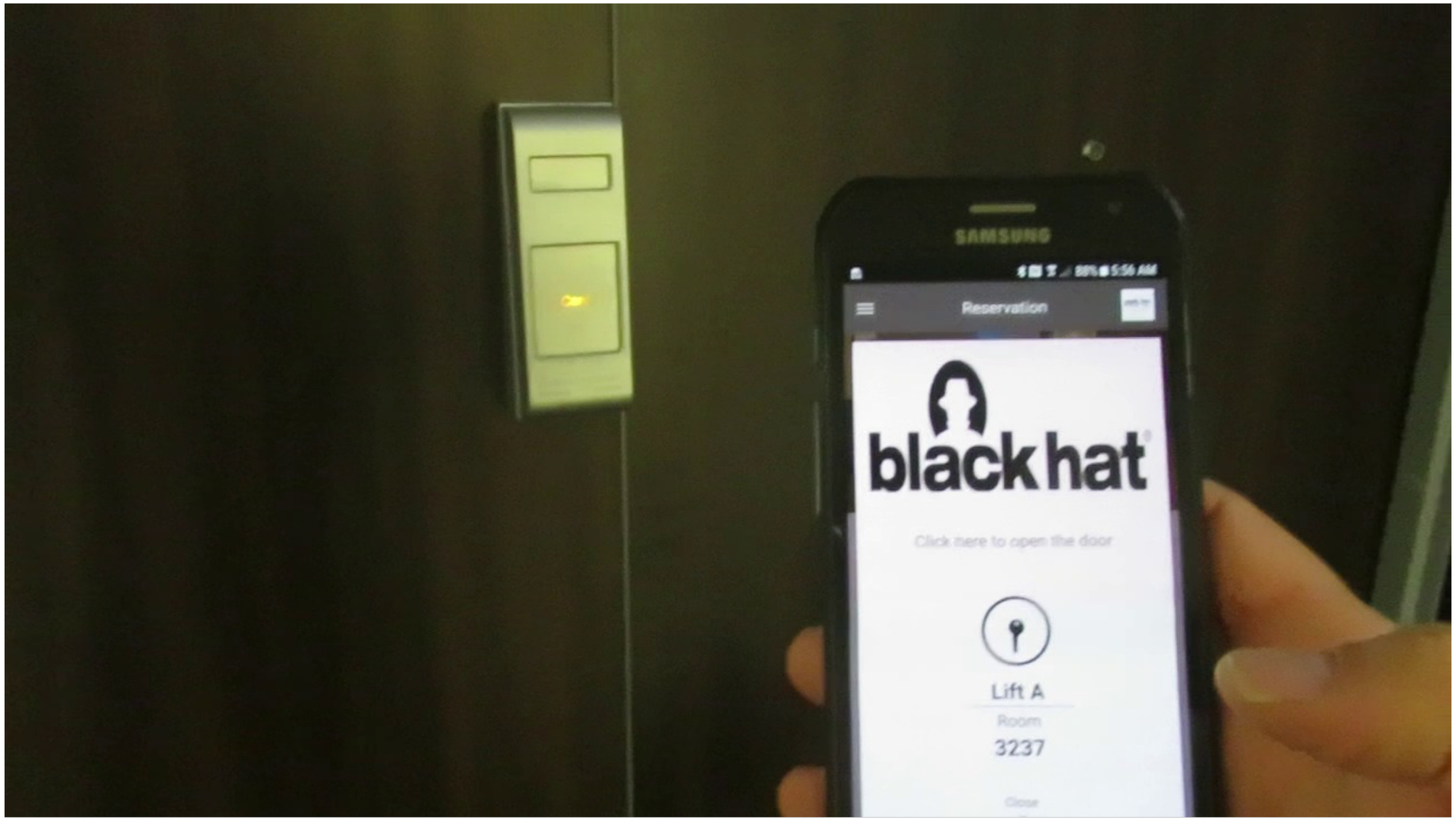
- Secure pairing not feasible
- Old hardware in locks, not always online
- Apps often made by 3rd parties, lock vendor just provides the SDK

Mobile Key

- Booking linked to app account, or added by user (sometimes using weak credentials)
- Online check-in
- Mobile key is transferred from backend to app

Mobile Key Demo

[Video that shows how to use the mobile app at a hotel door. (The graphics in the mobile app was modified using SSL MITM.)]



Video 1

Hotel “H”

Encrypted Mobile Key System

- The vendor has a secret key $\mathbf{K_s}$, known to the lock
- Backend to App: key **K** and encrypted key $\mathbf{K^*} = \mathrm{enc}_{\mathbf{Ks}}(K)$
- App to Lock: **K***
- Lock uses $\mathbf{K_s}$ to decrypt **K*** to **K**
- Key **K** now known to App and lock, but not to an eavesdropper; $\mathbf{K_s}$ still unknown to App
- Further BLE traffic is AES-encrypted with Key **K**

Encrypted Mobile Key System

- Didn’t find obvious attack vector,
  except for extracting $\mathbf{K_s}$ from the physical lock[1],
  which we haven't tried :)
- No further experiments, because on the second stay,
  the mobile key system was deactivated.

[1] cf. Thomas, Blackhat USA 2014: Reverse-Engineering the Supra iBox

Manufacturer “M”

Vulnerable System

- Found system early 2019 in an upper class hotel
- Mobile key used in elevator, rooms and fitness center
- Analyzed TLS and BLE traffic

Key from Backend

```
2019-07-25 03:23:08 GET https://app[redacted]'api/v1/devices/mobile_key/8f
                        dcc75e-a290-4633-9fb8-865c9472ba63
                        ← 200 OK application/json 702b 140ms
          Request                  Response                        Detail
X-Request-Id:                 48dd45a5-7610-4ba3-a684-f5853f5696dd
X-Runtime:                    0.047805
Strict-Transport-Security:    max-age=31536000; includeSubDomains
JSON                                                              [m:Auto]
{
    "device_token": "[redacted]",
    "exp_date": "2019-07-25 00:00:00.000",
    "key_type": [redacted],
    "mobile_key": {
        "da": "2019-07-25T14:00+00:00",
        "dt": [
            140,
            2,
            253,
            1,
            254,
            248,
[21/48]                                          ?:help q:back [*:21984]
```

Key from Backend

Data seen from Backend (TLS)

```
"dt": [
    140, = 0x8c
    2,   = 0x02
    253, = 0xfd
    1,   = 0x01
    254, = 0xfe
    248, = 0xf8
```

Data seen in HCI log (BLE)

```
Bluetooth HCI ACL Packet
Bluetooth L2CAP Protocol
Bluetooth Attribute Protocol
    Opcode: Write Request (0x12)
    Handle: 0x000e (Unknown: Unknown)
    Value: 300000000000050e18c02fd01fef8fdf9
    [Response in Frame: 622]
```

Full BLE Trace

```
Lock:   0000
Lock:   000103001ec05d6bb5190707051b2b19e0
App:    000102000012000101010101bbec98f3
Lock:   0001040104d612ffeafad012
App:    300000000000044ca8c02fd01fef8fdf9   = Key
App:    31605803e9196317fb5b9e8c6e616b7ba6    (all bytes from
App:    32ca06cfbc48c67697f0c34897948c218c    backend)
App:    33cf3f2a462f78d9c8874b6bb021b70034
Lock:   0002190707051b00090ca500000001af08
Lock:   0002
```

Note: The description was slightly modified to protect the innocent not yet patched devices.

Further Analysis

```
Lock:   0000
Lock:   000103001ec05d6bb5190707051b2b19e0   = Lock MAC,CRC
App:    000102000012000101010101bbec98f3     = App Nonce,CRC
Lock:   0001040104d612ffeafad012             = Lock Nonce,CRC
App:    300000000000044ca8c02fd01fef8fdf9    = Special CRC, Key
App:    31605803e9196317fb5b9e8c6e616b7ba6     (all bytes from
App:    32ca06cfbc48c67697f0c34897948c218c     backend)
App:    33cf3f2a462f78d9c8874b6bb021b70034
Lock:   0002190707051b00090ca500000001af08   = Lock confirmation: open
Lock:   0002
```

Note: The description was slightly modified to protect the innocent not yet patched devices.

CRC Reversing

- Tools for CRC reversing are available, e.g. CRC RevEng
- We just used a custom Python script and searched for CRC-16 parameters that matched in at least 2 messages, assuming the CRC is located at the end of a message

```
Trying different polynomials and start values...
Trying polynomial 0x2f15...
[...]
Trying polynomial 0x[redacted]...
Match found! Polynomial: 0x[redacted] Seed: 0x73 Final XOR: 0xffff
```

CRC Reversing

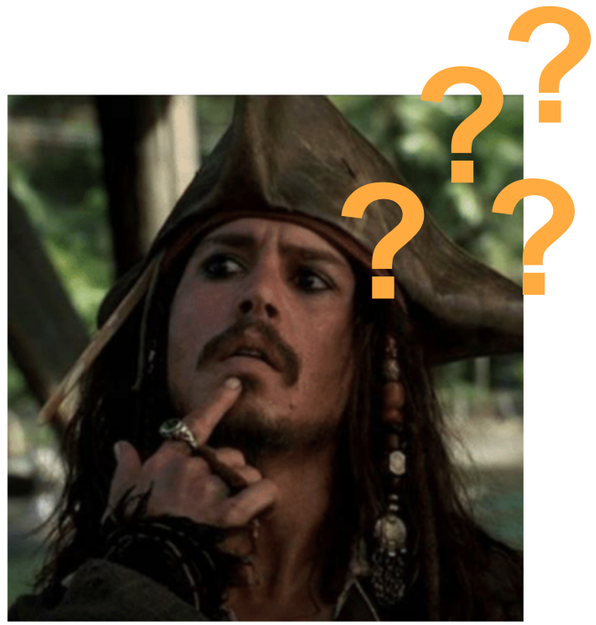
- Seed for CRC of first msg turned out to be a value received from the backend (“sc” / constant within hotel)
- Seed for CRC of next msg is CRC of previous msg
- But for the most important part, the credential packet, the CRC calculation was more complicated:

`00 00 00 00 00 00`	**`0c 3b`**	`8c 02 fd 01 fe`	`9e f2 3b`
6 bytes always zero	2 bytes **changing each session**	5 bytes constant per hotel	3 bytes constant per stay

Note: The description was slightly modified to protect the innocent not yet patched devices.

CRC Reversing

- So we had 1 block with the CRC obviously not at the end, some constant blocks, 6 zero bytes, and 16 changing bits
- And 3 CRC-16 values and 2 session nonces to play with…
- [... some playing around …]



Note: The description was slightly modified to protect the innocent not yet patched devices.

CRC Reversing

This intermediary byte sequence (and seed CRC3)

```
84 3c  45 f2  88 40  34 f1  8c 02 fd 01 fe 9e f2 3b
nonce1 CRC1   nonce2 CRC2
```

yields the final CRC-16 value **`0c3b`**.

→ Now we know how to create the credential packet:

```
00 00  00 00  00 00  0c 3b  8c 02 fd 01 fe 9e f2 3b
overwritten          CRC
with zeroes          inserted here
```

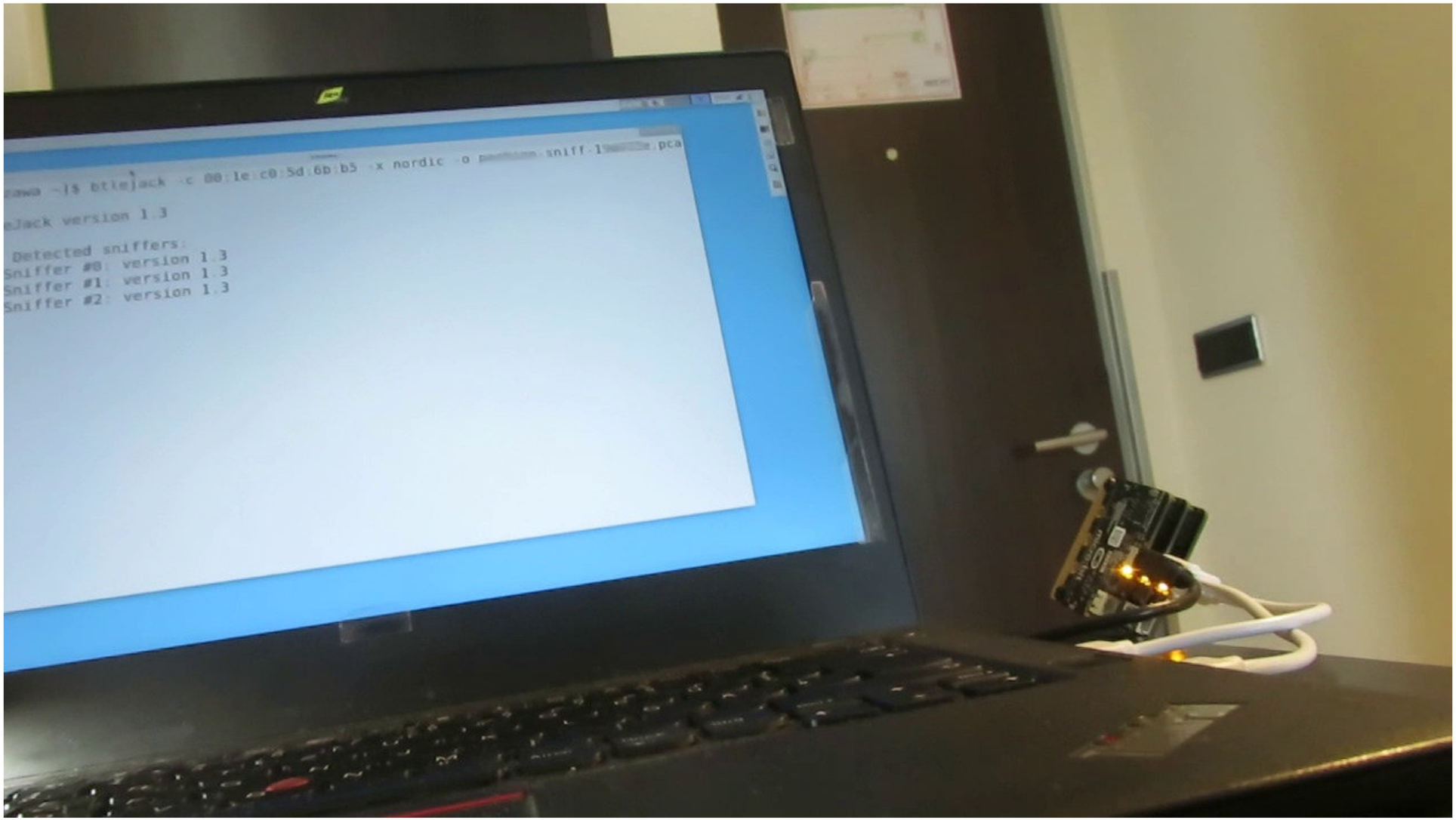
Note: The description was slightly modified to protect the innocent not yet patched devices.

Preparing an Attack

- Created a Python script
  - Input: Device name, credential bytes (as sniffed from previous opening)
  - Calculates CRCs, handles BLE communication (using bluepy)

Sniffing a Mobile Key

[Video that shows how BLE data is sniffed off the air.]



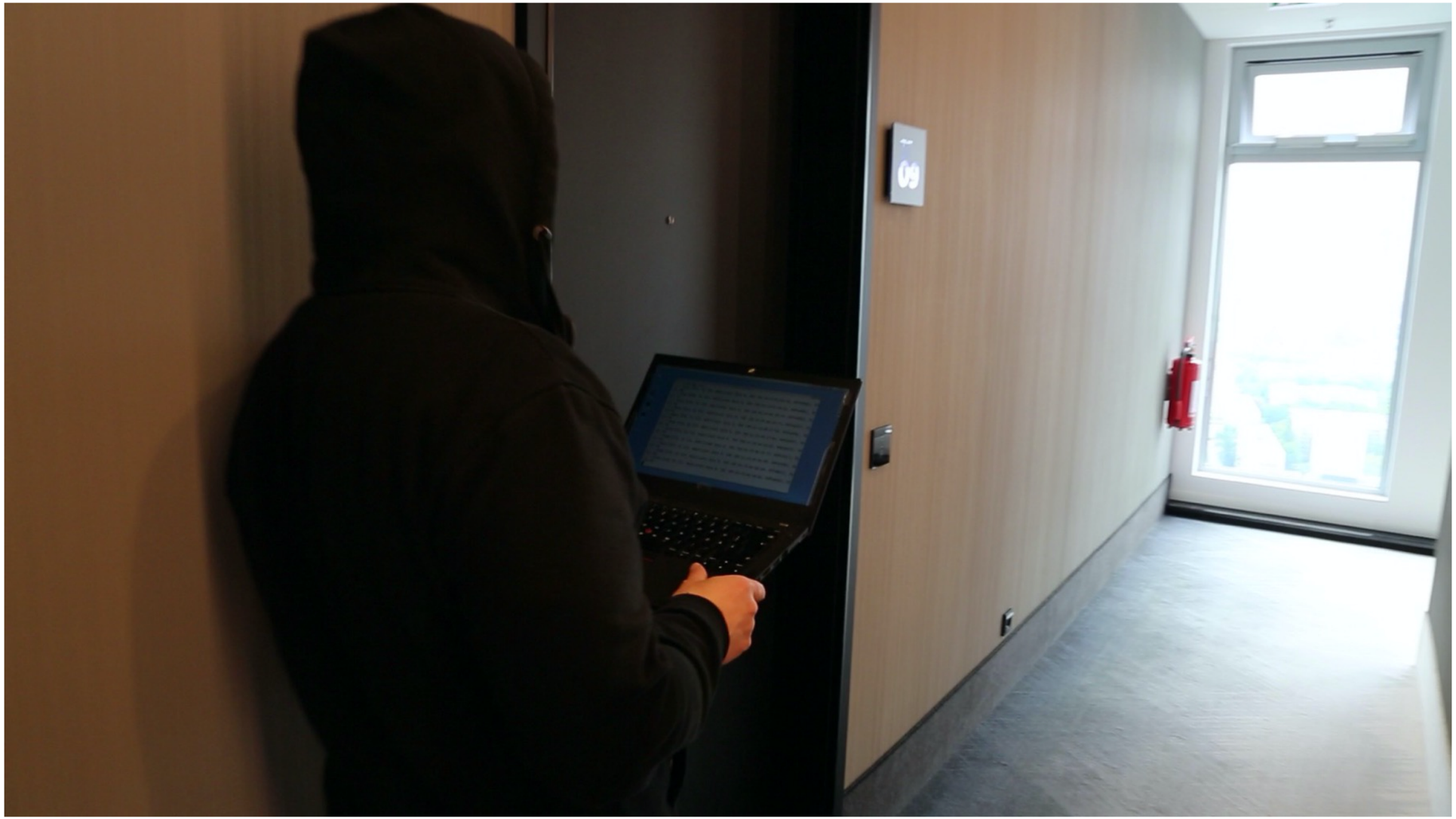
Video 2

Executing the Script

```
[root@zawa mmk-unlock-master]# python mmk-unlock.py AHPKUJzL 300000000000000381a8c02fd01fef
b5b9e8c6e616b7ba6 32ca06cfbc48c67697f0c34897948c218c 33cf3f2a462f78d9c8874b6bb021b70034
Derived from device name AHPKUJzL: SC == 115, Room Number == 3237
Extracted mobile key: 8c02fd01fef8fdf9605803e9196317fb5b9e8c6e616b7ba6ca06cfbc48c67697f0c3
8d9c8874b6bb021b70034
[*] scanning (3s)...
 [-] Room 3236, SC 115, Additional Data 0, 156 (00:1e:c0:5d:72:94, AHPKQJzb), RSSI=-88
 [-] Room 3237, SC 115, Additional Data 0, 156 (00:1e:c0:5d:6b:b5, AHPKUJzL), RSSI=-83
 [-] Room 3137, SC 115, Additional Data 0, 155 (00:1e:c0:5d:73:e8, AHPEEJuC), RSSI=-94
 [-] Room 3337, SC 115, Additional Data 0, 157 (00:1e:c0:4f:32:f3, AHPQkJ0Q), RSSI=-97
unlocking in progress...
[1] Connecting...
Initializing BLE peripheral class...
Setting the delegate...
MyDelegate registered
Discovering the BLE service...
Discovering the write characteristic...
```

Breaking into the Room

[Video that shows how Ray opens the hotel room door with sniffed data. (He opened doors only with permission of an authorized user, no actual Breaking In happened!)]



Video 3

Enjoy the View

Some more Scripting

- Created test target (also Python script)
  - simulates a lock
  - handles BLE communication in the peripheral role (using pybleno)
- Now we could play with this at home :)

How Big Is the Problem?

- Found more hotel chains using the product
- BLE names are easy to check on-site, without actual room booking
- After booking a room, we found an even simpler variation of the protocol deployed (the “final / special” CRC part is left out)

Responsible Disclosure

Disclosure Timeline

- 2019-04-18: First vendor notification, immediate response
- 2019-04-26: Technical details to vendor
- 2019-05-02: Vendor questions feasibility
- 2019-05-06: Proof of concept code sent
- 2019-05-29: Vendor acknowledges vulnerability
- 2019-06-28: Vendor discusses update plans

Update Plans and Challenges

- Locks in “our” first hotel are online, can be updated remotely
- Others need someone going from door to door with an update device
- Multiple app vendors have to integrate the new SDK
- Lesson learned: Identify all affected parties early

And Now for Something Completely Different

Do you remember this presentation?

The KABA MAS X-09™ High Security Safe Lock.

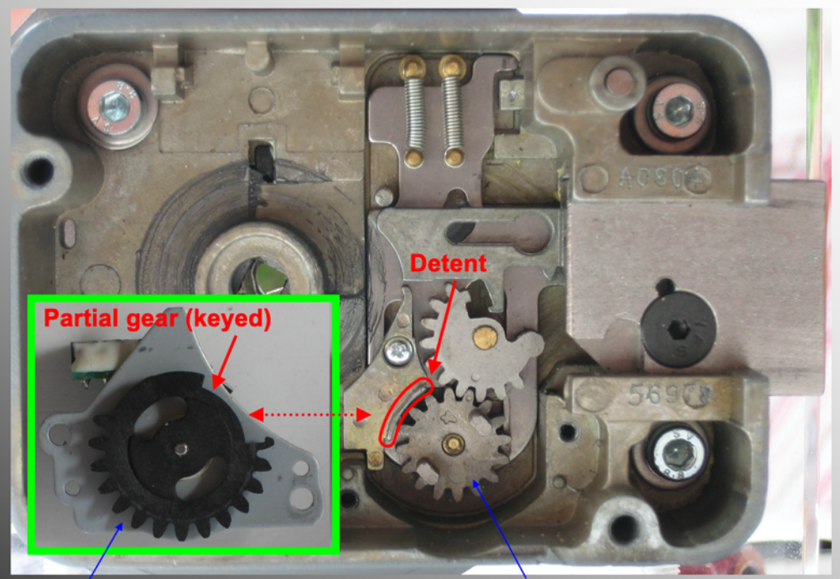
A Hands-On Presentation at LockCon 2008.

Sneek, The Netherlands, 10 Oct 2008, Michael U. Huebler.

TOOOL
The Open Organisation Of Lockpickers
LockCon 2008

LockCon 2008
X-09 Presentation by Michael U. Huebler
Page 10

The motor. Secure against bumping and vibration.



DEF CON 27 had a surprise…

REUTERS

Business Markets World Politics TV More

TECHNOLOGY NEWS AUGUST 6, 2019 / 9:58 PM / 3 MONTHS AGO

Exclusive: High-security locks for government and banks hacked by researcher

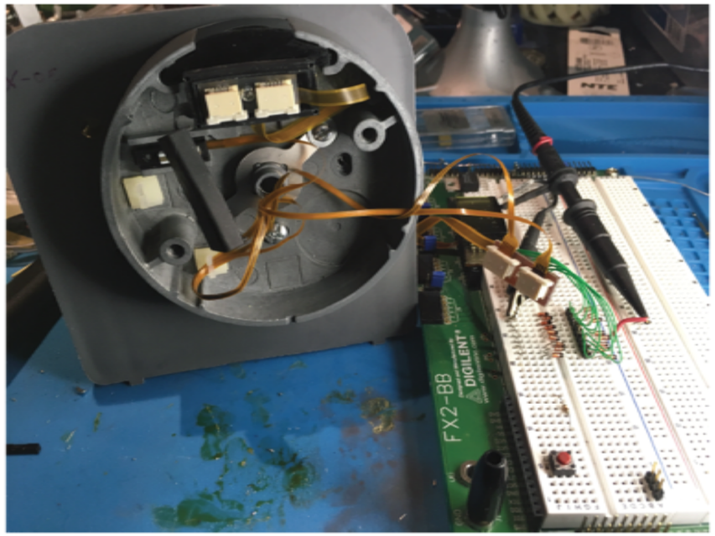
Joseph Menn

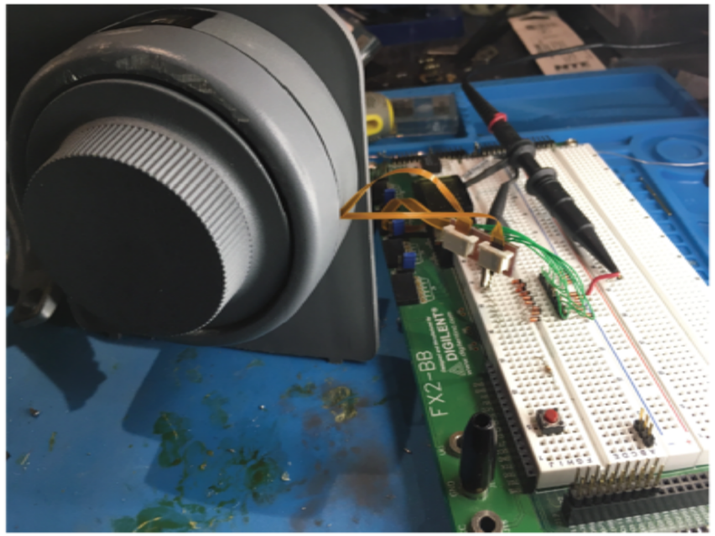
4 MIN READ

SAN FRANCISCO (Reuters) - Hackers could crack open high-security electronic locks by monitoring their power, allowing thieves to steal cash in automated teller machines, narcotics in pharmacies and government secrets, according to research to be presented Friday at the annual Def Con hacking conference in Las Vegas.

Mike Davis: “No MAS: (misadventures in high security lock design)”

IOActive
Hardware | Software | Wetware
SECURITY SERVICES





LockCon 2008

X-09 Presentation by Michael U. Huebler

Page 17

The electronic card (back cover assembly).

Thermal relocker activator

Serial number sticker

UV-active paint

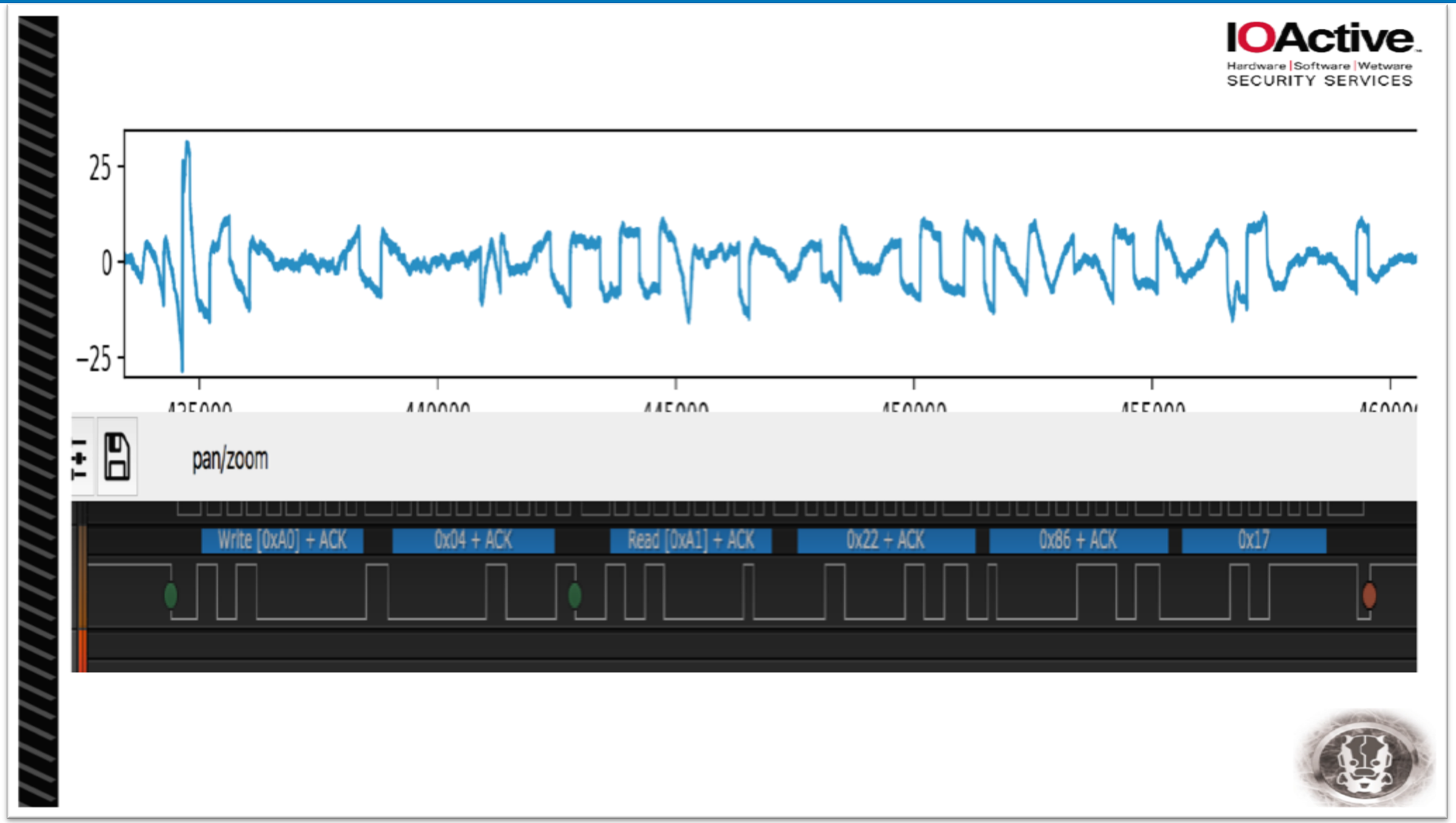
Processor (P87C52)

I^2C serial connection

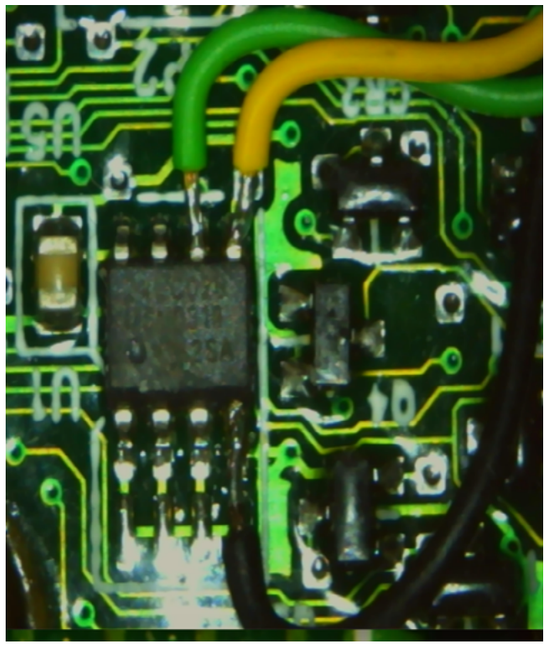
EEPROM (24LC02B)

LCD Driver

Signal on the “outside” LCD cables carries information about the internal EEPROM accesses



Easy to reproduce



However, the secret combination is not transmitted in “plain text”, but obfuscated

R/W	Addr	Value
Read	0x00	0x55
Read	0x01	0xAA
Read	0x04	0x58
Read	0x05	0x99
Read	0x06	0x53
Read	0x81	0x58
Read	0x82	0x99
Read	0x83	0x53
Read	0xFD	0x58
Read	0xFE	0x99
Read	0xFF	0x53
Write	0x0C	0x00

R/W	Addr	Value
Read	0x0A	0x00
Read	0x0E	0x00
Read	0x8A	0x00
Read	0x8B	0x00
Read	0x07	0x00

(long pause while the combination is dialed)

R/W	Addr	Value
Read	0x00	0x55
Read	0x01	0xAA
Read	0x0F	0x61
Read	0x10	0x84
Read	0x11	0x73
Read	0x12	0xE1
Read	0x13	0x90
Read	0x14	0x31
Read	0x15	0x7C
Read	0x16	0x3B
Read	0x17	0x40
Read	0x18	0x09
Read	0x19	0x3C
Read	0x1A	0x5B
Read	0x1B	0x43
Read	0x1C	0x90
Read	0x1D	0x49
Read	0x1E	0xE1
Read	0x1F	0x76
Read	0x20	0x61
Read	0x21	0x7C
Read	0x22	0x0B
Read	0x23	0x82
Read	0x24	0x90
Read	0x25	0x3C
Read	0x26	0x0D
Read	0x27	0x85
Read	0x28	0x51
Read	0x29	0xC1
Read	0x2A	0x92
Read	0x2B	0x30
Read	0x2C	0x35
Read	0x2D	0x44
Read	0x2E	0x1C
Read	0x2F	0x60
Read	0x30	0x81
Read	0x31	0x14

...

...

...

(Serial number of the X-09: 589953 – Combination: 12 34 56)

LOCKCON

Thanks for your attention!

Questions?

Contact: mh@tosl.org

Some Useful Links

BLE exploration tool for your smartphone:
https://apps.apple.com/app/lightblue-explorer/id557428110 /
https://play.google.com/store/apps/details?id=com.punchthrough.lightblueexplorer

Modifying Android app manifest to make app trust user CAs
https://medium.com/@elye.project/android-nougat-charlesing-ssl-network-efa0951e66de

Rebuild/Sign APK
https://gist.github.com/AwsafAlam/f53312cbb912cf3e4267a5971cd75db0

JADX decompiler:
https://github.com/skylot/jadx (Also can simply be done online: https://www.google.com/search?&q=online+jadx)

If you are interested in locks and lock picking:
https://toool.nl/Publications
http://lockpicking.org (German)