argXtract: Deriving IoT Security Configurations via Automated Static Analysis of Stripped ARM Cortex-M Binaries

Pallavi Sivakumaran, Jorge Blasco
Hackers Can Seize Control of Electric Skateboards and Toss Riders

Thousands of webcams vulnerable to attack

Smart camera and baby monitor

Warnings

Smartwatches Are a Security Nightmare

DDoS attack that disrupted internet may have the worst for security of

Kid's Smartwatches Are a Security Nightmare Despite Years of

Hacking risks lead to recall of 500,000

Hacking risk led to recall of 500,000

Packmakers due to patient death in

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from

Packmakers due to patient death from
Root Cause?

CVE-2015-2880: TRENDnet WiFi Baby Cam TV-IP743SIC has a password of admin for the backdoor root account

CVE-2017-17436: An issue was discovered in the software on Vaultek Gun Safe VT20i products. There is no encryption of the session between the Android application and the safe

CVE-2021-33220: An issue was discovered in CommScope Ruckus IoT Controller 1.7.1.0 and earlier. Hard-coded API Keys exist

>> configuration issues
Source of Configuration Data?

- Devices
- Mobile Apps
- Web Portals
- Firmware
Problems

- Firmware analysis is complex
- Firmware analysis for hub/gateway devices may try to identify familiar filesystem structures or standard functions, due to the use of Linux/VxWorks-based OSs
- *Peripheral* firmware analysis can be very complex
  - No/non-standard OS
  - The security-relevant configuration calls may be made from an application to an underlying technology stack, where the stack is often shipped separately from the application firmware
  - Tend to be available only as stripped binaries (no ELF headers/symbols/section information)
  - Increasingly featuring Cortex-M processors (Thumb instructions, often inline data)
uint8_t passkey[] = "123456";
ble_opt_t ble_opt;
ble_opt.gap_opt.passkey.p_passkey = &passkey[0];
err_code = sd_ble_opt_set(BLE_GAP_OPT_PASSKEY, &ble_opt);

DISASSEMBLY OF UNSTRIPPED BINARY

<table>
<thead>
<tr>
<th>Instruction address</th>
<th>Hex representation</th>
<th>Opcode</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>1eaba:</td>
<td>4ab8</td>
<td>ldr r2, [pc, #736]</td>
<td>(1ed9c)</td>
</tr>
<tr>
<td>1eabc:</td>
<td>ab06</td>
<td>add r3, sp, #24</td>
<td></td>
</tr>
<tr>
<td>1eabe:</td>
<td>6811</td>
<td>ldr r1, [r2, #0]</td>
<td></td>
</tr>
<tr>
<td>1eac0:</td>
<td>2022</td>
<td>movs r0, #34</td>
<td></td>
</tr>
<tr>
<td>1eac2:</td>
<td>9106</td>
<td>str r1, [sp, #24]</td>
<td></td>
</tr>
<tr>
<td>1eac4:</td>
<td>8891</td>
<td>ld rh r1, [r2, #4]</td>
<td></td>
</tr>
<tr>
<td>1eac6:</td>
<td>8099</td>
<td>strh r1, [r3, #4]</td>
<td></td>
</tr>
<tr>
<td>1eac8:</td>
<td>7992</td>
<td>ld rb r2, [r2, #6]</td>
<td></td>
</tr>
<tr>
<td>1eaca:</td>
<td>a908</td>
<td>add r1, sp, #32</td>
<td></td>
</tr>
<tr>
<td>1eacc:</td>
<td>719a</td>
<td>str r2, [r3, #6]</td>
<td></td>
</tr>
<tr>
<td>1ead0:</td>
<td>f7ffe3a</td>
<td>bl 1e748 &lt;sd_ble_opt_set&gt;</td>
<td></td>
</tr>
<tr>
<td>1ed9c:</td>
<td>00021f14</td>
<td>.word 0x00021f14</td>
<td></td>
</tr>
<tr>
<td>21f0c:</td>
<td>... 3231 3433 3635 0000</td>
<td>(%. .123456..</td>
<td></td>
</tr>
</tbody>
</table>

DISASSEMBLY OF STRIPPED BINARY

<table>
<thead>
<tr>
<th>Instruction address</th>
<th>Hex representation</th>
<th>Opcode</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x3aba:</td>
<td>b84a</td>
<td>ldr r2, [pc, #0x2e0]</td>
<td></td>
</tr>
<tr>
<td>0x3abc:</td>
<td>06ab</td>
<td>add r3, sp, #0x18</td>
<td></td>
</tr>
<tr>
<td>0x3abe:</td>
<td>1168</td>
<td>ldr r1, [r2]</td>
<td></td>
</tr>
<tr>
<td>0x3ac0:</td>
<td>2220</td>
<td>movs r0, #0x22</td>
<td></td>
</tr>
<tr>
<td>0x3ac2:</td>
<td>0691</td>
<td>str r1, [sp, #0x18]</td>
<td></td>
</tr>
<tr>
<td>0x3ac4:</td>
<td>9188</td>
<td>ld rh r1, [r2, #4]</td>
<td></td>
</tr>
<tr>
<td>0x3ac6:</td>
<td>9980</td>
<td>strh r1, [r3, #4]</td>
<td></td>
</tr>
<tr>
<td>0x3ac8:</td>
<td>9279</td>
<td>ld rb r2, [r2, #6]</td>
<td></td>
</tr>
<tr>
<td>0x3aca:</td>
<td>08a9</td>
<td>add r1, sp, #0x20</td>
<td></td>
</tr>
<tr>
<td>0x3acc:</td>
<td>9a71</td>
<td>str r2, [r3, #6]</td>
<td></td>
</tr>
<tr>
<td>0x3ace:</td>
<td>0893</td>
<td>str r3, [sp, #0x20]</td>
<td></td>
</tr>
<tr>
<td>0x3ad0:</td>
<td>fff73afe</td>
<td>bl #0x3748</td>
<td></td>
</tr>
</tbody>
</table>

Incorrect instruction addresses
1. Identify app code base using Vector Table and self-targeting branches.

2. Identify inline data using Reset Handler, LDR, table branches.

3. Estimate function blocks using Vector Table, BL, exit bypasses.

4. Pinpoint instructions of interest as SVCalls or function calls.

5. Generate trace chains and trace through assembly keeping track of registers/RAM.

6. Process register/memory objects according to Argument Definition Objects.

>> Ultimately extract arguments to (security-relevant) functions of interest

https://github.com/projectbtle/argXtract
1. Identify app code base

<table>
<thead>
<tr>
<th>Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRQn</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>IRQ2</td>
</tr>
<tr>
<td>IRQ1</td>
</tr>
<tr>
<td>IRQ0</td>
</tr>
<tr>
<td>Systick</td>
</tr>
<tr>
<td>PendSV</td>
</tr>
<tr>
<td>Reserved</td>
</tr>
<tr>
<td>Reserved for Debug</td>
</tr>
<tr>
<td>SVCall</td>
</tr>
</tbody>
</table>

```
00021a70 <NMI_Handler>:
  21a70: e7fe  b.n21a70 <NMI_Handler>

00021a72 <HardFault_Handler>:
  21a72: e7fe  b.n21a72 <HardFault_Handler>
```

```
0x6a70: fee7  b  #0x6a70
0x6a72: fee7  b  #0x6a72
```
1. Identify app code base

- Identify the app code base:
  \[ \text{UNSTRIPPED BINARY} \]
  
  - 0x6a70:
    - fee7 b #0x6a70
  - 0x6a72:
    - fee7 b #0x6a72

- From the application vector table:
  \[ \text{Offset (h)} \]
  
  - 0x21a70 - 0x6a70 = \text{0x1b000} \]

- Self-targeting branch addresses within firmware:
  
  - 0x21a70 - 0x6a70 = \text{0x1b000}

- The address 0x1b000 is from the application vector table and represents the app code base.
2. Identify inline data

**LDR and variants**

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1b8a2</td>
<td>ldr</td>
<td>r3, [pc, #0x20] ; 0x1b8c4</td>
</tr>
<tr>
<td>0x1b8c4</td>
<td>movs</td>
<td>r1, #0x6c</td>
</tr>
<tr>
<td>0x1b8c6</td>
<td>movs</td>
<td>r0, #0</td>
</tr>
</tbody>
</table>

**TBB and TBH**

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2a6e0</td>
<td>cmp</td>
<td>r3, #4</td>
</tr>
<tr>
<td>0x2a6e2</td>
<td>bhi.n</td>
<td>2a740 &lt;characteristic_add+0x16c&gt;</td>
</tr>
<tr>
<td>0x2a6e4</td>
<td>tbb</td>
<td>[pc, r3]</td>
</tr>
<tr>
<td>0x2a6e8</td>
<td>ldmibhi</td>
<td>sp, {r0, r2, r4, r7, r8, fp, ip, pc}</td>
</tr>
<tr>
<td>0x2a6ec</td>
<td>;</td>
<td>&lt;UNDEFINED&gt; instruction: 0x23220029</td>
</tr>
</tbody>
</table>

**__ARM_common_switch8 calls**

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1ca5e</td>
<td>bl</td>
<td>1b4f4 __ARM_common_switch8&gt;</td>
</tr>
<tr>
<td>0x1ca62</td>
<td>ldrne</td>
<td>r0, [r5, #-3352] ; 0xffffffff2e8</td>
</tr>
<tr>
<td>0x1ca66</td>
<td>smmlarvs</td>
<td>r5, r7, r7, r4</td>
</tr>
<tr>
<td>0x1ca6a</td>
<td>ldmdahi</td>
<td>pc!, {r2, r3, r4, r5, r6, fp, pc}^ ; &lt;UNPREDICTABLE&gt;</td>
</tr>
<tr>
<td>0x1ca6e</td>
<td>mrc1s</td>
<td>14, 4, sp, cr14, cr1, {4}</td>
</tr>
<tr>
<td>0x1ca72</td>
<td>ldmg1t</td>
<td>lr, {r1, r2, r3, r4, r7, r9, s1, fp, ip, pc}</td>
</tr>
<tr>
<td>0x1ca76</td>
<td>atnle</td>
<td>&lt;illegal precision&gt;m f5, #0.5</td>
</tr>
<tr>
<td>0x1ca7a</td>
<td>ldrdcs</td>
<td>r1, [pc, -1r]</td>
</tr>
<tr>
<td>0x1ca7e</td>
<td>ldr</td>
<td>r2, [sp, #8]</td>
</tr>
</tbody>
</table>
3. Estimate function blocks

*Function start addresses* = \{0x21A70, 0x21A72, 0x21A74, 0x21A76, 0x21A78, 0x21A7A\}

Known function start addresses
3. Estimate function blocks

*Function start addresses* = \{0x1B198, 0x1B1C4, 0x1E87C, 0x21A70, 0x21A72, 0x21A74, 0x21A76, 0x21A78, 0x21A7A\}

**High-certainty function starts [bl-targets]**

<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1B160:</td>
<td>480c</td>
<td><code>ldr r0, [pc, #48];</code></td>
</tr>
<tr>
<td>1B162:</td>
<td>e000</td>
<td><code>b.n 1B166</code></td>
</tr>
<tr>
<td>1B164:</td>
<td>bf00</td>
<td><code>nop</code></td>
</tr>
<tr>
<td>1B166:</td>
<td>f000 f82d</td>
<td><code>bl 1B1C4</code></td>
</tr>
<tr>
<td>1B16A:</td>
<td>0020</td>
<td><code>movs r0, r4</code></td>
</tr>
<tr>
<td>1B16C:</td>
<td>0029</td>
<td><code>movs r1, r5</code></td>
</tr>
<tr>
<td>1B16E:</td>
<td>f003 fb85</td>
<td><code>bl 1E87C</code></td>
</tr>
<tr>
<td>1B172:</td>
<td>f000 f811</td>
<td><code>bl 1B198</code></td>
</tr>
<tr>
<td>1B176:</td>
<td>46c0</td>
<td><code>nop ; (mov r8, r8)</code></td>
</tr>
</tbody>
</table>
3. Estimate function blocks

*Function start addresses = {0x1B198, 0x1B1BC, 0x1B1C4, 0x1E87C, 0x21A70, 0x21A72, 0x21A74, 0x21A76, 0x21A78, 0x21A7A}*
4. Pinpoint instructions of interest

```c
uint8_t passkey[] = "123456";
ble_opt_t ble_opt;
ble_opt.gap_opt.passkey.p_passkey = &passkey[0];
err_code = sd_ble_opt_set(BLE_GAP_OPT_PASSKEY, &ble_opt);
```
4. Pinpoint instructions of interest

- ARM supervisor calls
  - Easy to identify within disassembly (instruction with opcode `svc`)
  - `svc` numbers of interest can be found within vendor SDKs

- Function calls
  - argXtract uses *function pattern files* to identify functions of interest, i.e., files representing expected outputs at specific addresses for given inputs
  - Function pattern file is passed to the set of identified functions. Each function is executed with the inputs in the function pattern file, and the outputs are compared with the expected values
  - Assumes that the function of interest produces unique, distinguishable outputs or artefacts
5. Generate trace chains and trace through assembly

- Register and memory contents are stored and processed during the trace
- Upon reaching the function/svc of interest, register+memory contents are returned for analysis
6. Process register/memory objects

Trace object

```json
{"sd_ble_opt_set": {
    'memory': {
        '20007f60': '31',
        '20007f61': '32',
        '20007f62': '33',
        '20007f63': '34',
        '20007f64': '35',
        '20007f65': '36',
        '20007f66': '00',
        '20007f68': '60',
        '20007f69': '7f',
        '20007f6a': '00',
        '20007f6b': '20',
        '20007f6c': '01',
        ...
    },
    'registers': {
        'pc': '0001ea96',
        'sp': '20007f48',
        'r0': '00000022',
        'r1': '20007f68',
        ...
    }
}}
```

Argument definition object

```json
{"args": {
    "0": {...},
    "1": {
        "in_out": "in",
        "ptr_val": "pointer",
        "data": {
            "p_opt": {
                "ptr_val": "pointer",
                "length_bits": 48,
                "type": "hex"
            }
        }
    }
}}
```

Output

```json
"output": {
    "sd_ble_opt_set": [{
        "opt_id": 34,
        "p_opt": "313233343536"
    }]
}
```

Hex for "123456"
Evaluation: Test Set

- No existing ground truth for Cortex-M
- Manually generated test set
- 28 binaries
- Chipset vendors: NXP, STMicroelectronics, Nordic Semiconductor and Texas Instruments
- Technologies: Zigbee, ANT, BLE, Thread and 802.15.4
- Compilers: GCC, IAR, Keil and Clang (depending on options made available by chipset vendors)
## Evaluation: Function Identification

<table>
<thead>
<tr>
<th>Bin File</th>
<th>argXtract</th>
<th>radare2</th>
<th>ghidra</th>
<th>Bin File</th>
<th>argXtract</th>
<th>radare2</th>
<th>ghidra</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ID</td>
<td>#Fns</td>
<td>TPR</td>
<td>EFPR</td>
<td>TPR</td>
<td>EFPR</td>
<td>TPR</td>
</tr>
<tr>
<td></td>
<td>0a1</td>
<td>324</td>
<td>100</td>
<td>0.29</td>
<td>95.68</td>
<td>2.19</td>
<td>87.96</td>
</tr>
<tr>
<td></td>
<td>1d7</td>
<td>951</td>
<td>93.27</td>
<td>0.97</td>
<td>74.24</td>
<td>3</td>
<td>73.08</td>
</tr>
<tr>
<td></td>
<td>443</td>
<td>598</td>
<td>100</td>
<td>0</td>
<td>83.78</td>
<td>3.05</td>
<td>83.95</td>
</tr>
<tr>
<td></td>
<td>589</td>
<td>1486</td>
<td>97.51</td>
<td>0.68</td>
<td>83.24</td>
<td>1.59</td>
<td>84.79</td>
</tr>
<tr>
<td></td>
<td>646</td>
<td>166</td>
<td>98.8</td>
<td>0</td>
<td>80.72</td>
<td>0.73</td>
<td>77.71</td>
</tr>
<tr>
<td></td>
<td>681</td>
<td>1961</td>
<td>97.86</td>
<td>0.56</td>
<td>94.19</td>
<td>0.7</td>
<td>87.51</td>
</tr>
<tr>
<td></td>
<td>70b</td>
<td>115</td>
<td>95.65</td>
<td>0</td>
<td>67.83</td>
<td>0</td>
<td>73.04</td>
</tr>
<tr>
<td></td>
<td>928</td>
<td>520</td>
<td>95.38</td>
<td>0.93</td>
<td>90.19</td>
<td>0</td>
<td>71.15</td>
</tr>
<tr>
<td></td>
<td>989</td>
<td>762</td>
<td>95.8</td>
<td>9.7*</td>
<td>69.27</td>
<td>8.62</td>
<td>63.1</td>
</tr>
<tr>
<td></td>
<td>bad</td>
<td>839</td>
<td>92.25</td>
<td>0.79</td>
<td>69.85</td>
<td>5.43</td>
<td>68.53</td>
</tr>
<tr>
<td></td>
<td>cb5</td>
<td>92</td>
<td>94.57</td>
<td>1.11</td>
<td>61.96</td>
<td>0</td>
<td>67.39</td>
</tr>
<tr>
<td></td>
<td>dd9</td>
<td>801</td>
<td>96.63</td>
<td>6.9*</td>
<td>95.93</td>
<td>6.36</td>
<td>88.15</td>
</tr>
<tr>
<td></td>
<td>e2d</td>
<td>698</td>
<td>96.42</td>
<td>0.35</td>
<td>94.99</td>
<td>0</td>
<td>86.25</td>
</tr>
<tr>
<td></td>
<td>f37</td>
<td>1585</td>
<td>95.21</td>
<td>1.16</td>
<td>78.23</td>
<td>3.18</td>
<td>78.36</td>
</tr>
</tbody>
</table>
## Case Studies

<table>
<thead>
<tr>
<th>Case Study</th>
<th>Chipset Vendor</th>
<th>Technology</th>
<th>Config Mechanism</th>
<th>Tested Security Metrics</th>
<th>Tested Calls of Interest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nordic BLE (243)</td>
<td>Nordic Semiconductor</td>
<td>Bluetooth Low Energy</td>
<td>Supervisor calls</td>
<td>Security of BLE Data</td>
<td>sd_ble_gatts_characteristic_add, sd_ble_gatts_service_add</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Use of Fixed Passkeys</td>
<td>sd_ble_opt_set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>User Tracking due to Fixed Addresses</td>
<td>sd_ble_gap_address_set, sd_ble_gap_addr_set, sd_ble_gap_privacy_set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Manufacturer/Device Names and Privacy</td>
<td>sd_ble_gap_device_name_set</td>
</tr>
<tr>
<td>BlueNRG (2)</td>
<td>STMicroelectronics</td>
<td>Bluetooth Low Energy</td>
<td>Function calls</td>
<td>BLE Address Privacy</td>
<td>aci_hal_write_config_data, aci_gap_init</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BLE Pairing Security</td>
<td>aci_gap_set_io_capability, aci_gap_set_authentication_requirement, aci_gap_set_authorization_requirement</td>
</tr>
<tr>
<td>ANT (9)</td>
<td>Nordic Semiconductor</td>
<td>ANT</td>
<td>Supervisor calls</td>
<td>ANT Channel Security</td>
<td>sd_ant_enable</td>
</tr>
</tbody>
</table>
Findings

- Over 50% of binaries had little to no protection for vendor-defined data (including ECG monitors, HID devices, smart watches, safety aids...)
- Inconsistent application of permissions to restrict access to data
- Use of fixed pairing passkeys, i.e., weakened pairing mechanism
- 95% of devices – including all wearables – use fixed device addresses, enabling device (and potentially also user) tracking
Conclusions

- Framework to extract security configurations from stripped Cortex-M binaries
- Several vulnerabilities identified in real-world IoT devices
- Still work to be done: improving function pattern matching
- Ground truth!
Thank You