Le tabac, c’est tabou, on en viendra tous à bout!

*Bière Sécu Toulouse*

November 21 2019
Samuel (@w4kfu) CHEVET
Agenda

1. Introduction
2. The wrong way
3. The right way
4. Conclusion
A co-worker asked me to look at the firmware of its e-cigarette

Sounds like a fun late night home project

Target: Minikin
Fun fact

All chips seems brushed: no more free information
Firmware

- Firmware files available on vendor website
- Less than 80 KB
- Upgrade tool: Windows executable

```bash
$ binwalk -Y firmware

<table>
<thead>
<tr>
<th>DECIMAL</th>
<th>HEXADECIMAL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>56103</td>
<td>0x10237</td>
<td>ARM executable code, 16-bit (Thumb), big endian, at least 643 valid instructions</td>
</tr>
</tbody>
</table>
```
Firmware

- Trying to disassemble the firmware with the ARM CPU detected leads to garbage ...

```
binwalk -E firmware
```
Agenda

1. Introduction
2. The wrong way
3. The right way
4. Conclusion
Upgrade tool

- BoxedApp packer
- Full-fledged applications converted into a single executable
- ‘Emulate’ file system and a system registry
### .main section

<table>
<thead>
<tr>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORIGINAL_EXECUTABLE.EXE</td>
<td></td>
</tr>
<tr>
<td>EXE_PackerStub32.dll</td>
<td>bxpck section handler</td>
</tr>
<tr>
<td>BoxedAppSDK.dll</td>
<td>Virtual Environment / FileSystem / Registry</td>
</tr>
<tr>
<td>BoxedAppSDK_AppDomainManager.dll</td>
<td></td>
</tr>
<tr>
<td>BoxedAppSDKThunk32.dll</td>
<td></td>
</tr>
<tr>
<td>TLSSupport32.dll</td>
<td>.NET manager</td>
</tr>
</tbody>
</table>
.bxpck section

Series of different entry type

- 0x00 : LENGTH [DWORD]
- 0x04 : TYPE [WORD]

- 0x02 : Command line overriding
- 0x03 : Import section information
- 0x05 : Virtual file entry
- 0x06 : Start ROOT
- 0x07 : Virtual registry sub-entry
- 0x08 : Virtual registry entry
- 0x0A : End ROOT
Upgrade tool

- Virtual file container contains 6 uninteresting DLLs:
  - hidapi.dll
  - libgcc_s_dw2-1.dll
  - mingwm10.dll
  - msvcr100d.dll
  - qtcore4.dll
  - qtgui4.dll

- Application allows to put the device in DFU mode and send the firmware image without any treatment 😊
Agenda

1. Introduction
2. The wrong way
3. The right way
4. Conclusion
Guess 100

- Looking at the file under an hex editor
- Values seems stored in Little-Endian

![Hexdump Image]

- Signature?
- 0x13B9C = file_size (0x13C00) - 0x64
- HIWORD identical...
We are probably dealing with some ARM Cortex

Based on the vector table entries (pattern of 4 bytes), is the ROM start address: 0x08000000?
Vector table

ROM

- Create ROM section
- ROM start address: 0x08000000
- ROM size: 0x13C00

Input file

- Loading address: 0x08000000
- File offset: 0x20

CODE16

<table>
<thead>
<tr>
<th>ROM: 08000000</th>
<th>CODE16</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM: 08000000</td>
<td>DCD</td>
<td>0x200036CF Initial SP Value</td>
</tr>
<tr>
<td>ROM: 08000004</td>
<td>DCD</td>
<td>0x8001B3E Reset Vector</td>
</tr>
<tr>
<td>ROM: 08000008</td>
<td>DCD</td>
<td>0x800D5F2 NMI Vector</td>
</tr>
<tr>
<td>ROM: 0800000C</td>
<td>DCD</td>
<td>0x800DA4 Hard Fault</td>
</tr>
<tr>
<td>ROM: 08000010</td>
<td>DCD</td>
<td>0x800DA2 Memory Management Fault</td>
</tr>
<tr>
<td>ROM: 08000014</td>
<td>DCD</td>
<td>0x800DA0 Bus Fault</td>
</tr>
<tr>
<td>ROM: 08000018</td>
<td>DCD</td>
<td>0x800DSBE Usage Fault</td>
</tr>
<tr>
<td>ROM: 0800001C</td>
<td>DCD</td>
<td>0x26EF RESERVED</td>
</tr>
<tr>
<td>ROM: 08000020</td>
<td>DCD</td>
<td>0x565C RESERVED</td>
</tr>
<tr>
<td>ROM: 08000024</td>
<td>DCD</td>
<td>0x565C RESERVED</td>
</tr>
<tr>
<td>ROM: 08000028</td>
<td>DCD</td>
<td>0x565C RESERVED</td>
</tr>
</tbody>
</table>
## Problems

- Initial SP Value not aligned?
- Reserved vector not NULL?
Grep

- Grepping the sequence of bytes \texttt{EF 26 00 00}

- \texttt{0x000026EF} repeated 8 times
- The other reserved vector (\texttt{0x0000565C}) repeated 8 times
- ...
Cryptography?

- Sequence of bytes seems to have lead us to ZERO pages
- We are probably dealing with a XOR cipher
- Each key entry seems only on 16 bits

**Key format?**

+ 0x00 : key_00  [DWORD] * 0x08
+ 0x20 : key_01  [DWORD] * 0x08
+ 0x40 : key_02  [DWORD] * 0x08
...
Identifying key length

Key length: 0x800 bytes
Full key extraction

- Firmware doesn’t seem to contain ZERO page of 0x800 bytes
- 3 different versions of the firmware available

Solution

1. Split firmware in block of 0x800 bytes
2. Compute the number of occurrence of every 0x04 bytes repeated every 0x20 bytes
3. Repeat the task for all firmwares available
Wait ...strings

**Firmware_1.out**

<table>
<thead>
<tr>
<th>Offset (h)</th>
<th>00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F</th>
</tr>
</thead>
<tbody>
<tr>
<td>000070A0</td>
<td>FE F7 6B B8 B4 02 00 20 B0 23 01 08 9F 5A 01 08</td>
</tr>
<tr>
<td>000070B0</td>
<td>5D 43 01 08 53 48 4F 52 54 00 00 00 41 54 4F 4D</td>
</tr>
<tr>
<td>000070C0</td>
<td>49 5A 45 52 00 00 00 00 48 49 47 48 00 00 00 00</td>
</tr>
<tr>
<td>000070D0</td>
<td>54 45 4D 50 45 52 00 00 57 4F 52 4B 00 00 00 00</td>
</tr>
<tr>
<td>000070E0</td>
<td>4F 56 45 52 54 49 4D 45 00 00 00 4C 4F 57 00</td>
</tr>
</tbody>
</table>

(0x74C0 % 0x800) / 0x20 = 0x26; Key index broken

**Firmware_2.out**

<table>
<thead>
<tr>
<th>Offset (h)</th>
<th>00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F</th>
</tr>
</thead>
<tbody>
<tr>
<td>000074B0</td>
<td>64 07 00 20 03 6F 01 08 B9 52 01 08 OB 6D 01 08</td>
</tr>
<tr>
<td>000074C0</td>
<td>A3 48 4F 52 A4 00 00 00 B1 54 4F 4D B9 5A 45 52</td>
</tr>
<tr>
<td>000074D0</td>
<td>F0 00 00 00 B8 49 47 48 F0 00 00 00 A4 45 4D 50</td>
</tr>
<tr>
<td>000074E0</td>
<td>45 52 00 00 57 4F 52 4B 00 00 00 4F 56 45 52</td>
</tr>
</tbody>
</table>

(0x74C0 % 0x800) / 0x20 = 0x26; Key index broken
Wait ...bindiff

Firmware_1.out

Firmware_2.out

\[(0xC440 \% 0x800) / 0x20 = 0x22; \text{Key index broken}\]
Full key extraction

- After some adjustments we have the full key
  - Index 30, 34, 38, 60, 62 were broken
After some adjustments we have the full key

- Index 30, 34, 38, 60, 62 were broken

Some key are on 32 bits ...wat?

- keys[54] = 0xD6ADBCEF
- keys[58] = 0xDEADBEEF
- keys[63] = 0xE3ADBEEF
Last step

- Loading the ROM at address 0x08000000 doesn’t seem to work
- Vector table entry points to nonsense disassembly

Solution 1

- Grepping the disassembly for something that looks like a Reset_Handler

| ROM:080001CE | 00 00 | MOVs | R0, R0 |
| ROM:080001D0 | 09 48 | LDR  | R0, =(sub_800F41A+1) |
| ROM:080001D2 | 80 47 | BLX  | R0     ; sub_800F41A |
| ROM:080001D4 | 09 48 | LDR  | R0, =(loc_8003CF4+1) |
| ROM:080001D6 | 00 47 | BX   | R0     ; loc_8003CF4 |

- Compute the delta between the vector table entry address and the address found
Last step

- Loading the ROM at address 0x08000000 doesn’t seem to work
- Vector table entry points to nonsense disassembly

Solution 2

- Grepping the immediate value of Vector Table Offset Register: 0xE000ED08

```
ROM:0800B854  2E 49  LDR       R1, =0xE000ED08
ROM:0800B856  2D 48  LDR       R0, =0x8003C00
ROM:0800B858  08 60  STR       R0, [R1]

...  

ROM:0800B90C  08 3C+dword_800B90C  DCD 0x8003C00
ROM:0800B910  08 ED+off 800B910  DCD 0xE000ED08
```
Agenda

1. Introduction
2. The wrong way
3. The right way
4. Conclusion
Conclusion

- The BoxedApp executable is useless in our case
- The firmware key can be extracted from only observation
- Generation of the XOR cipher key seems broken (32 bits values sometimes)
- My co-worker can now reverse engineer it without problems!
QUESTIONS?