POSITIVE TECHNOLOGIES

# Blackboxing Diebold-Nixdorf ATMs

**Vladimir Kononovich**
Senior ICS Security Specialist

**Alexei Stennikov**
Independent Researcher

ptsecurity.com

# Who are we?

## Vladimir Kononovich:

- Reverse-engineering (since 2008)
- Romhacking (my hobby)
- Writing tools for IDA/Ghidra
- Ghidra ideologist

# Who are we?

## Alexei Stennikov:

- Hardware expert
- ICS/SCADA security researcher
- ATM/POS security researcher
- Some skills of RE

# ATM hardware internals

- Less-secure upper part
- Safe-zone (lower part)

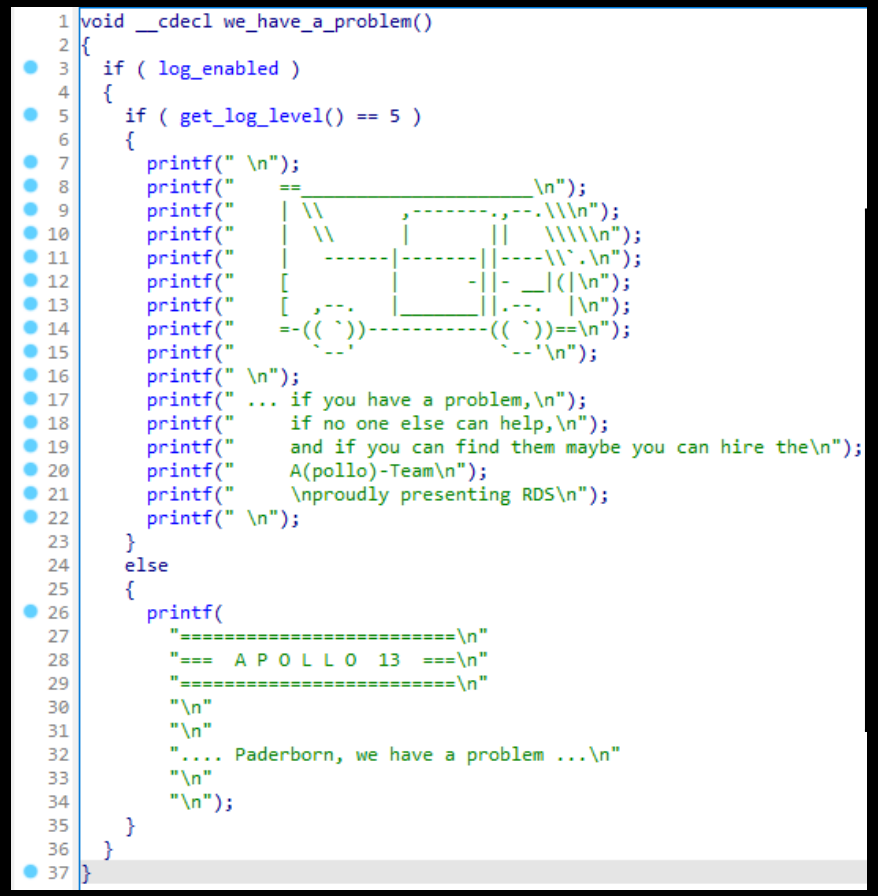
Safe-zone includes a dispenser controller

# Our previous talk at hw.io

Our
hardwear.io 2018
talk (youtube)
https://www.youtube.com/watch?v=L5yl4A1npVU

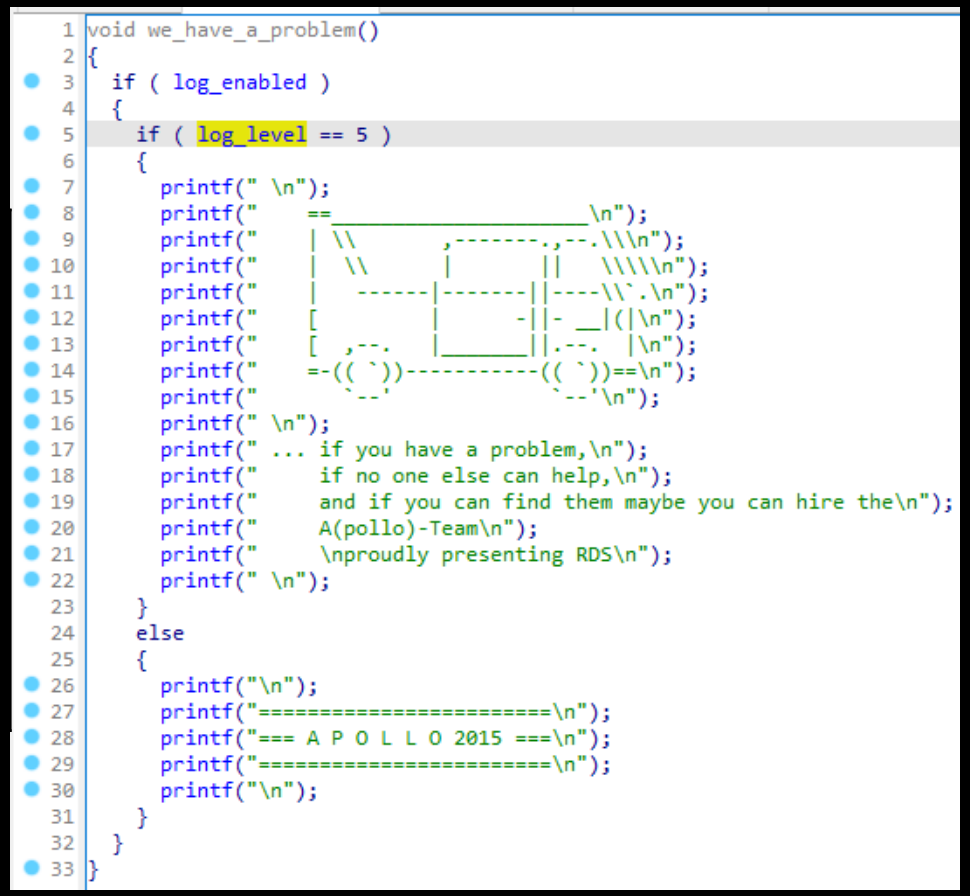
- ATM internals
- ATM attacks types
- What is Blackbox attack?
- NCR dispensers vulnerability

# Paderborn, we have a problem



```
void __cdecl we_have_a_problem()
{
  if ( log_enabled )
  {
    if ( get_log_level() == 5 )
    {
      printf(" \n");
      printf("    ==_____________________\n");
      printf("    | \\       ,-------.,--.\\\\n");
      printf("    |  \\      |       ||   \\\\\\n");
      printf("    |   ------|-------||----\\\`.\n");
      printf("    [         |       -||- __|(|\n");
      printf("    [  ,--.   |_______||.--.  |\n");
      printf("    =-(( `))----------(( `))==\n");
      printf("       `--'              `--'\n");
      printf(" \n");
      printf(" ... if you have a problem,\n");
      printf("     if no one else can help,\n");
      printf("     and if you can find them maybe you can hire the\n");
      printf("     A(pollo)-Team\n");
      printf("     \nproudly presenting RDS\n");
      printf(" \n");
    }
    else
    {
      printf(
        "========================\n"
        "===  A P O L L O  13  ===\n"
        "========================\n"
        "\n"
        "\n"
        ".... Paderborn, we have a problem ...\n"
        "\n"
        "\n");
    }
  }
}
```

- FW downgrade
- Modified FW uploading
- SmartCard DoS *“feature”*
- Encryption bypass
- Withdrawal

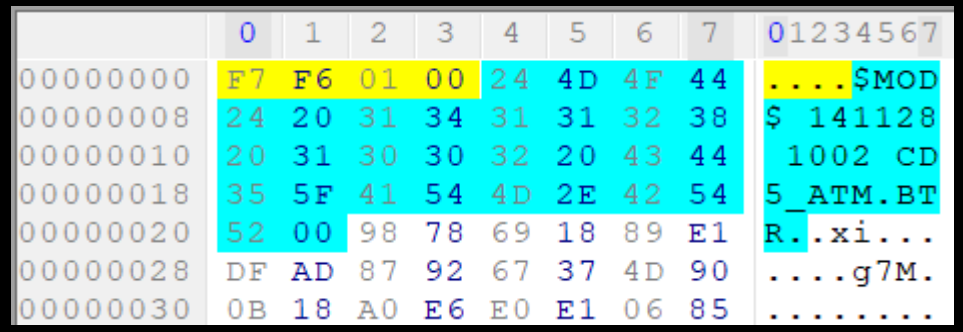


```
void we_have_a_problem()
{
  if ( log_enabled )
  {
    if ( log_level == 5 )
    {
      printf(" \n");
      printf("    ==_____________________\n");
      printf("    | \\       ,-------.,--.\\\\n");
      printf("    |  \\      |       ||   \\\\\\n");
      printf("    |   ------|-------||----\\\`.\n");
      printf("    [         |       -||- __|(|\n");
      printf("    [  ,--.   |_______||.--.  |\n");
      printf("    =-(( `))----------(( `))==\n");
      printf("       `--'              `--'\n");
      printf(" \n");
      printf(" ... if you have a problem,\n");
      printf("     if no one else can help,\n");
      printf("     and if you can find them maybe you can hire the\n");
      printf("     A(pollo)-Team\n");
      printf("     \nproudly presenting RDS\n");
      printf(" \n");
    }
    else
    {
      printf("\n");
      printf("========================\n");
      printf("=== A P O L L O 2015 ===\n");
      printf("========================\n");
      printf("\n");
    }
  }
}
```

# RM3/CMDv5 firmware files

## Parts:

- BTR (*bootloader*)
- FRM (*main firmware*)

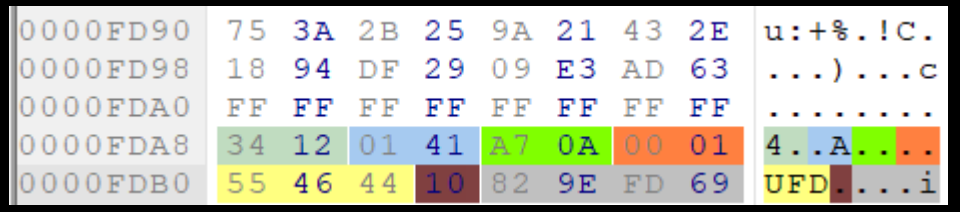


| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 01234567 |
|---|---|---|---|---|---|---|---|---|---|
| 00000000 | F7 | F6 | 01 | 00 | 24 | 4D | 4F | 44 | ....$MOD |
| 00000008 | 24 | 20 | 31 | 34 | 31 | 31 | 32 | 38 | $ 141128 |
| 00000010 | 20 | 31 | 30 | 30 | 32 | 20 | 43 | 44 | 1002 CD |
| 00000018 | 35 | 5F | 41 | 54 | 4D | 2E | 42 | 54 | 5_ATM.BT |
| 00000020 | 52 | 00 | 98 | 78 | 69 | 18 | 89 | E1 | R.xi... |
| 00000028 | DF | AD | 87 | 92 | 67 | 37 | 4D | 90 | ....g7M. |
| 00000030 | 0B | 18 | A0 | E6 | E0 | E1 | 06 | 85 | ..... |

- **Some size**
- **Firmware part name**

## Files:

- RM3_CRS.BTR / CD5_ATM.BTR
- RM3_CRS.FRM / CD5_ATM.FRM



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0000FD90 | 75 | 3A | 2B | 25 | 9A | 21 | 43 | 2E | u:+%.!C. |
| 0000FD98 | 18 | 94 | DF | 29 | 09 | E3 | AD | 63 | ...)...c |
| 0000FDA0 | FF | FF | FF | FF | FF | FF | FF | FF | ..... |
| 0000FDA8 | 34 | 12 | 01 | 41 | A7 | 0A | 00 | 01 | 4..A.... |
| 0000FDB0 | 55 | 46 | 44 | 10 | 82 | 9E | FD | 69 | UFD....i |

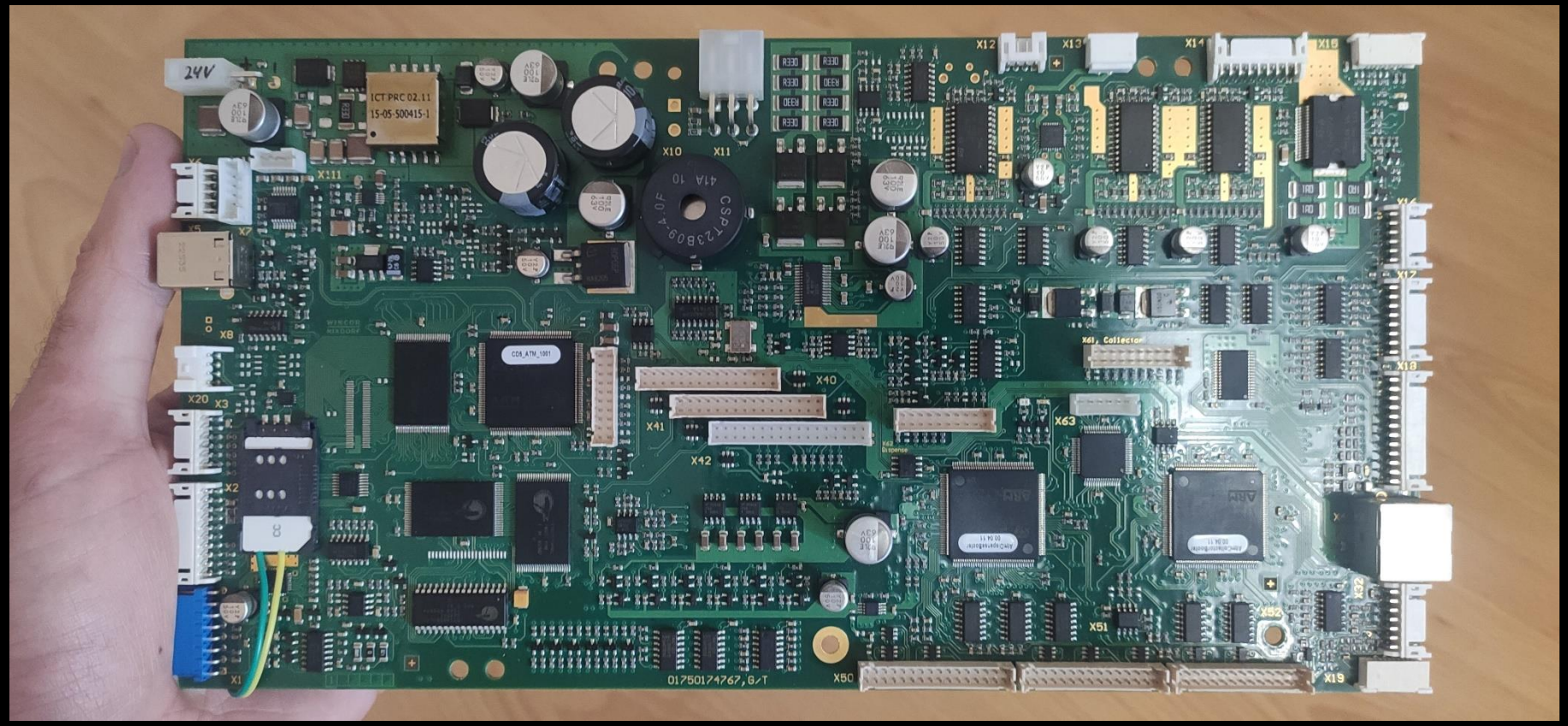
- **Device id**
- **Product id**
- **Vendor id**
- **?**
- **“UFD”**
- **?**
- **CRC32**

The rest is encrypted. No chance to decrypt.

Thank you for watching! Bye:)

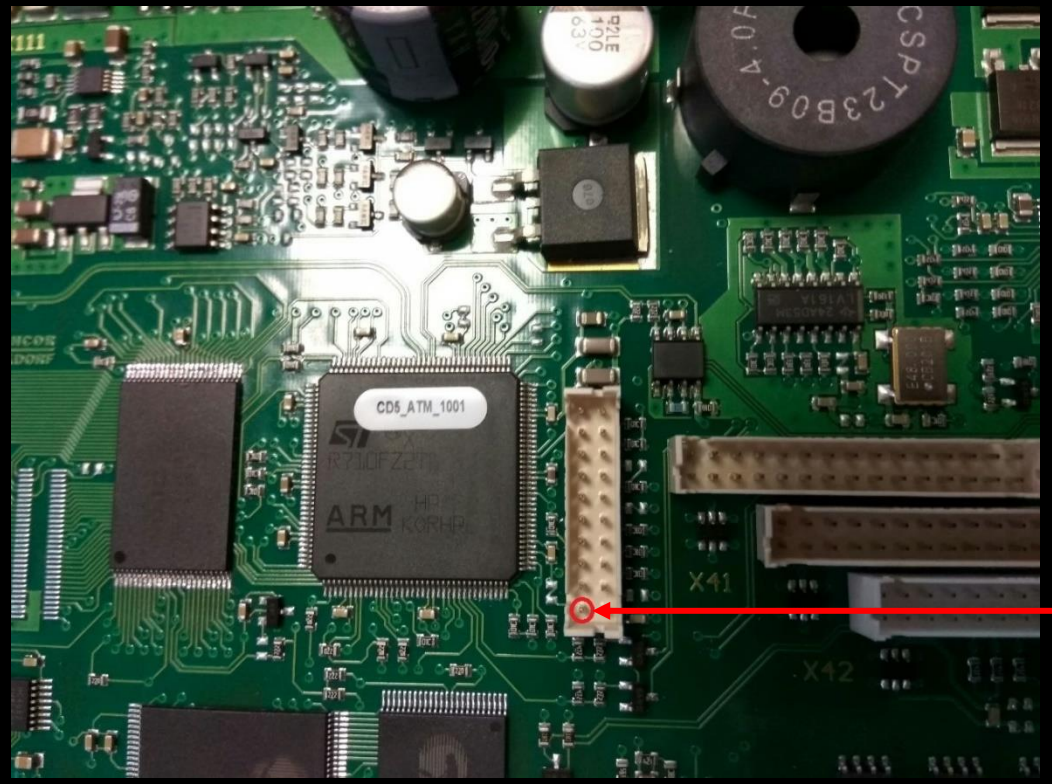
# But wait...

eBay can help us! Again…



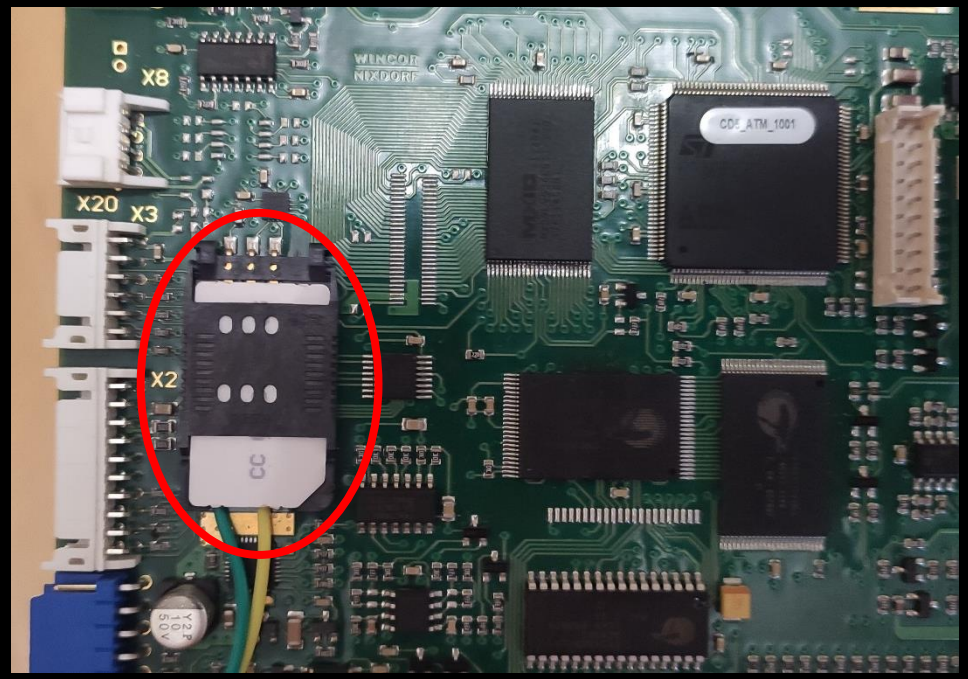
# Demo

# JTAG: Identifying connector & pins




| | | | | |
|---|---|---|---|---|
| 1 | • VREF | | • VSUPPLY | 2 |
| 3 | • nRST | | • GND | 4 |
| 5 | • TDI | | • GND | 6 |
| 7 | • TMS | | • GND | 8 |
| 9 | • TCK | | • GND | 10 |
| 11 | • RTCK | | • GND | 12 |
| 13 | • TDO | | • GND | 14 |
| 15 | • nRST | | • GND | 16 |
| 17 | • DBGRQ | | • GND | 18 |
| 19 | • DGBACK | | • GND | 20 |

# Another interesting place: Smartcard

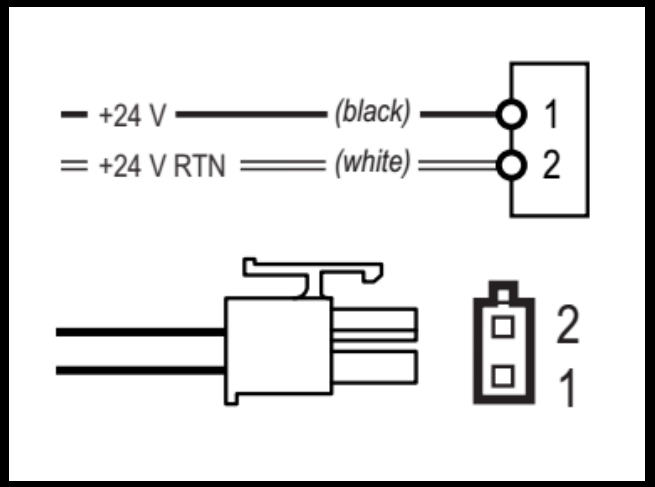



- USB encryption keys generation
- Session numbers/keys storage
- Different counters
- Certificates storage

Other “features”:)

- A whole system DoS

# Powering and testing
# FW uploading

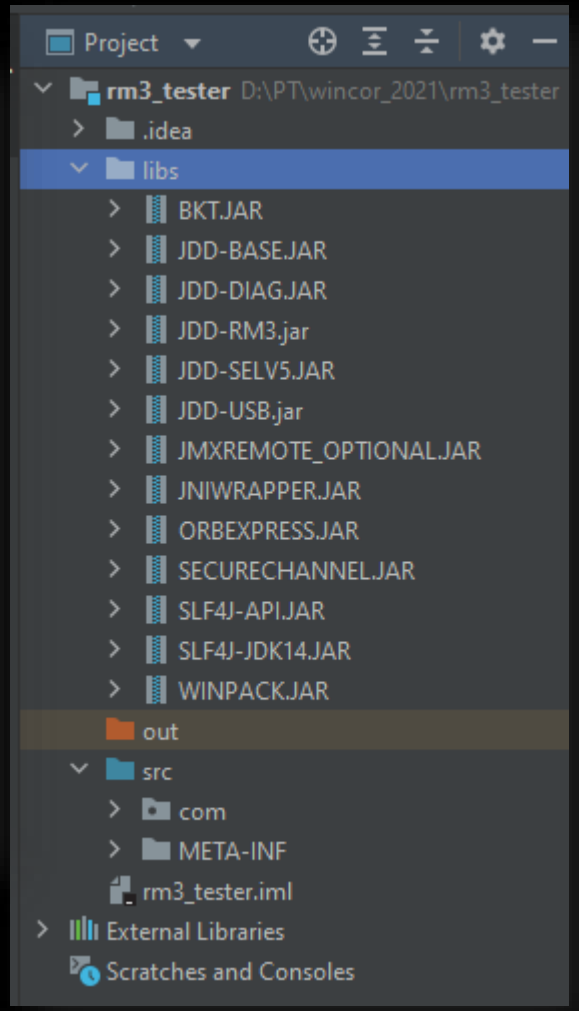



```java
public static void main(String[] args) {
    List<File> files = new ArrayList<File>();
    files.add(new File(args[0]));

    String[] fileTab = new String[files.size()];
    for (int i = 0; i < files.size(); ++i) {
        File file = files.get(i);
        if (file == null || !file.exists()) {
            System.out.println("One or more files do not exist");
            return;
        }
        fileTab[i] = file.getAbsolutePath();
    }

    Dfux dfux = new Dfux(VID, PID);

    try {
        dfux.download(fileTab, DfuxLoaderCtl.ALWAYS);
    } catch (DfuxException e) {
        e.printStackTrace();
    }
}
```



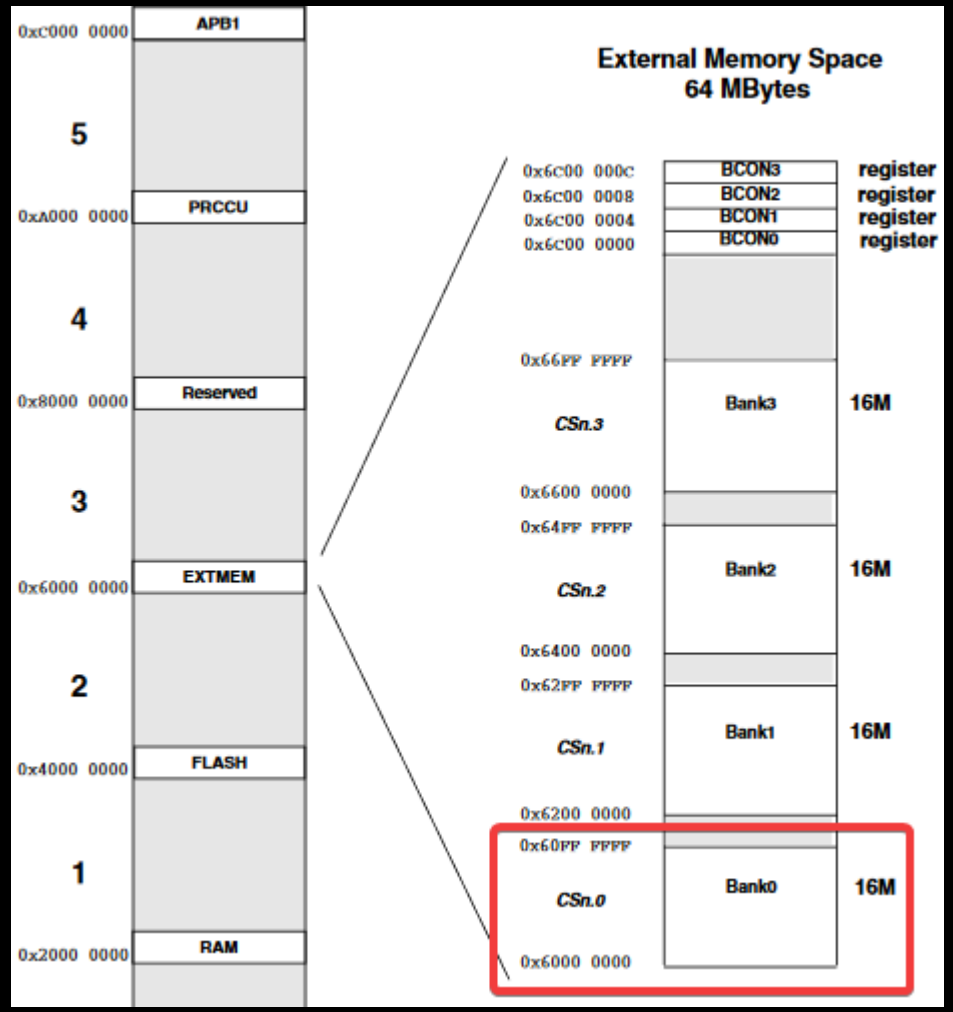

- + USB connection
- + Java-based software
  (easy to decompile and modify)

# Firmware dumping (CMDv5)

- Main CPU: STM STR710FZ2T6
- Image base: 0x60000000

## Two other CPUs:

- CollectorBooter: STR730FZ2T6
- DispenseBooter: STR730FZ2T6



# Firmware analysis (CMDv5)

Encryption algo – XTEA mod.
DELTA: 0xF27716BA. Rounds: 32

Init:

1. Read 5 LE-dwords after a $MOD$ name (header-dwords, HD)
2. key[n] = KEY1[n] ^ HD[n]; // where n: 0..3
3. data[0] = KEY0[0] ^ HD[0] ^ HD[1];
   data[1] = KEY0[1] ^ HD[2] ^ HD[3];

– *KEY0* and *KEY1* are unknown yet!

# Firmware analysis (CMDv5)

Decryption algo XTEA (Python):

```python
def decrypt_buffer(xtea_ctx, buf, start_offset, xor=0):
    offset = start_offset
    end_offset = len(buf)

    decrypted = ''
    while offset < end_offset:
        x1, x2 = unpack_from('<II', buf, offset)
        if x1 == 0xFFFFFFFF and x2 == 0xFFFFFFFF:
            break
        decrypted += xtea_ctx.crypt(buf, offset, xor=xor)
        offset += 8

    return decrypted
```

*Our python implementation*

```python
def dw(dd):
    return dd & 0xFFFFFFFF

def crypt(self, buf, offset, xor=0):
    v0 = self.data[0]
    v1 = self.data[1]

    self.data[1] = self.dw(self.data[1] + 1)

    s = 0
    for i in range(self.ROUNDS):
        v0 = self.dw(v0 + self.dw(self.dw(self.dw(self.dw(v1 << 4) ^ self.dw(v1 >> 5)) + v1) ^
                                  self.dw(s + self.key[s & 3])))
        s = self.dw(s + self.DELTA)
        v1 = self.dw(v1 + self.dw(self.dw(self.dw(self.dw(v0 << 4) ^ self.dw(v0 >> 5)) + v0) ^
                                  self.dw(s + self.key[self.dw(s >> 11) & 3])))

    x1, x2 = unpack_from('<II', buf, offset)
    x1 = self.dw(x1 ^ v0)
    x2 = self.dw(x2 ^ v1)

    return pack('<II', x1 ^ xor, x2 ^ xor)
```

# Firmware analysis (CMDv5)

## Decryption result:

- Sequential APLib archives (have AP32 header)
- Ends with 0xFFFFFFFFs

| 7 | 8 | 9 | A | B | C | D | E | F | 0123456789ABCDEF |
|---|---|---|---|---|---|---|---|---|---|
| 00 | A8 | 45 | 00 | 00 | 90 | 1A | F9 | 93 | AP32.....E..... |
| F4 | 1C | 15 | F0 | 9F | E5 | 04 | B8 | 04 | ....]M.......... |
| 78 | E1 | 02 | 7C | 24 | 80 | 48 | 84 | 91 | ...`p..x..\|$.H.. |
| 19 | A0 | E3 | CE | 12 | 81 | E2 | 1F | 67 | ."XDl.........g |
| FF | 89 | EA | 04 | C0 | 2D | E5 | 70 | 54 | ............-.pT |
| 08 | 05 | F0 | A0 | E1 | 0A | 14 | 48 | 0C | ......P.......H. |
| EF | 91 | EE | 63 | 38 | 9D | E4 | A0 | E0 | f.Hw......c8.... |
| 06 | 10 | 4F | E1 | 02 | 58 | F0 | 20 | 41 | N.._8-...O..X. A |
| 20 | 1E | BD | E8 | 01 | 07 | 6F | E1 | FF | ..0C0.. .....o.. |
| 64 | C0 | 65 | 12 | F8 | 88 | 4C | 80 | A8 | ...`...d.e...L.. |
| 81 | 40 | 5C | 04 | 16 | 01 | A0 | 11 | 04 | ...!)...@\...... |

| 0000FD28 | 91 | B2 | 82 | C2 | 69 | B1 | 7F | 33 | ....i..3 |
|---|---|---|---|---|---|---|---|---|---|
| 0000FD30 | B2 | 7C | F0 | 80 | E0 | FF | 86 | 92 | .\|...... |
| 0000FD38 | B1 | F3 | 33 | B2 | 49 | B1 | 26 | 7F | ..3.I.&. |
| 0000FD40 | B2 | 6C | 70 | A1 | D2 | 38 | 34 | B2 | .lp..84. |
| 0000FD48 | 97 | B1 | 99 | 99 | B2 | 41 | A4 | 85 | .....A.. |
| 0000FD50 | B2 | 47 | C1 | 0F | 7F | 6E | B1 | 24 | .G...n.$ |
| 0000FD58 | A3 | CE | 98 | 00 | FF | FF | FF | FF | .... .... |
| 0000FD60 | | | | | | | | | |

- Unpacked firmware

```
0123456789ABCDEF0123456789ABCDEF
................................
....`..`x..`|..`...`...`...`X..`
l..`.........................-.T...
....P.........-.H...............
..............N.._-...O...-. ...
 0............o............d...`
...`...`...d....................
................................
................................
@\............Zt....$MOD$ 141128
 1002 CD5_ATM.BTR................
5...4.....}....-.x.?*...L:......
)...,..!...!..,.+U>8A........f?3
./.2...0...1..*?.R...:.(O..5(|.9
g..0r.8-]Q.8..u..8..Y....p\*-C..
.iF(...%C},?../3.`...L...\...+.2
......Z.}.....:.H.b0...)?....1p&
```

# Firmware analysis (CMDv5)

## KEY0 and KEY1:

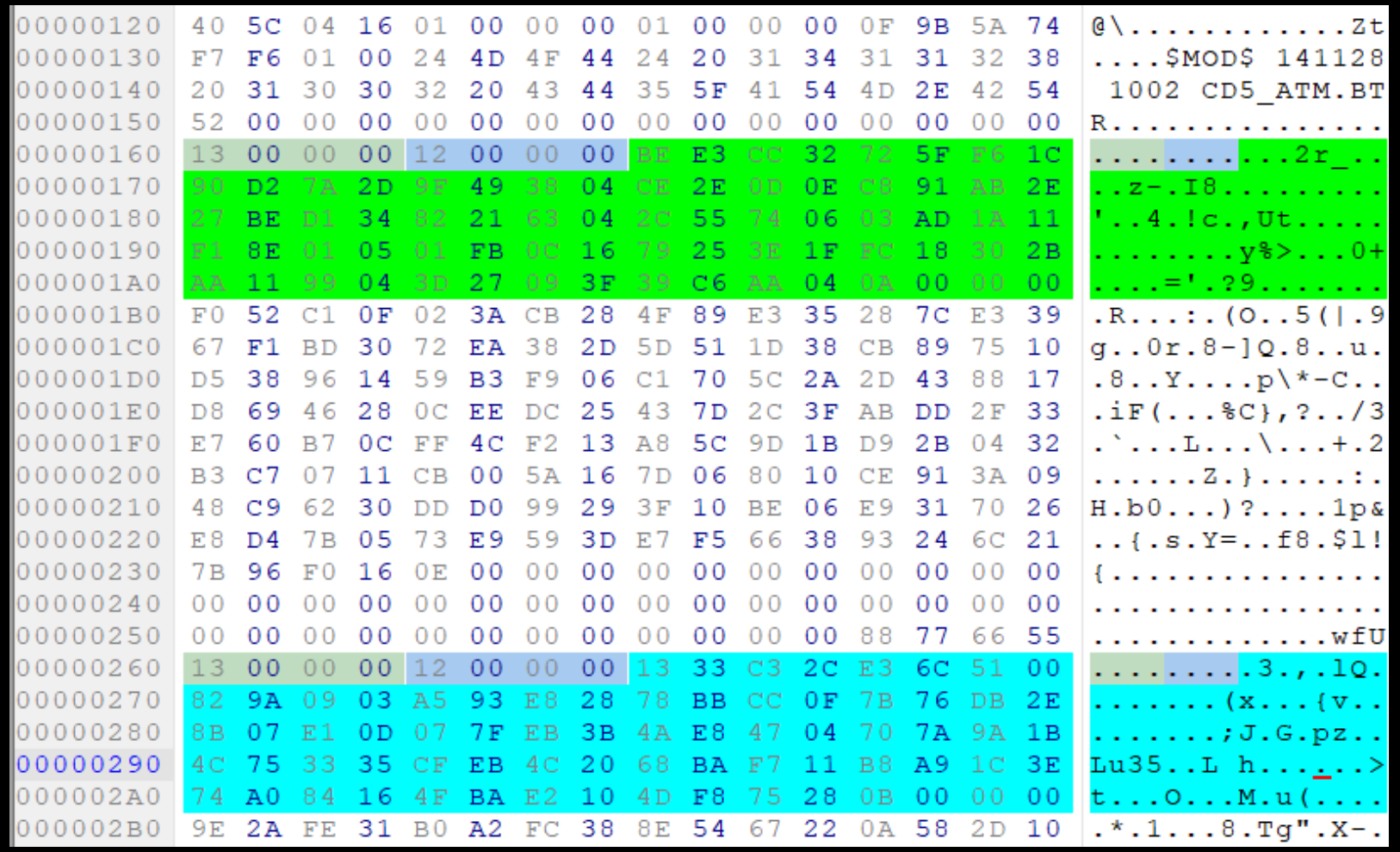
- Hardcoded! (base offset: 0x64000000)
- Ability to use OLD or ZEROed keys!

```
// unxoring dword to check against 0xDEAD signature
v16 = ((frm_buf[header_start + 0x16] << 16) |
       (frm_buf[header_start + 0x17] << 24) |
       (frm_buf[header_start + 0x15] << 8) |
       frm_buf[header_start + 0x14]) ^ v1->header_5_dwords[0];
if ( v16 >> 16 << 16 == 0xDEAD0000 )
{
  // pointing to old key0 (or to 0x00 filled area)
  xx1 = (int *)&old_keys0_block[8 * (unsigned __int8)v16];
  // pointing to old key1 (or to 0x00 filled area)
  xx2 = (int *)&old_keys1_block[16
                              * (((frm_buf[header_start + 0x16] << 16) |
                                 (frm_buf[header_start + 0x17] << 24) |
                                 (frm_buf[header_start + 0x15] << 8) |
                                 frm_buf[header_start + 0x14]) ^ *header_buf) & 0xFFF];
}
// Forming two "data"-dwords
// which will be used as XTEA encrypted data
dest2_xx[0] = *xx1;
dest2_xx[1] = xx1[1];
```

# Firmware analysis (CMDv5)

## Self-signing (bad practice)

- 0x160 – sign
- 0x260 – modulus
- 0x360 – data

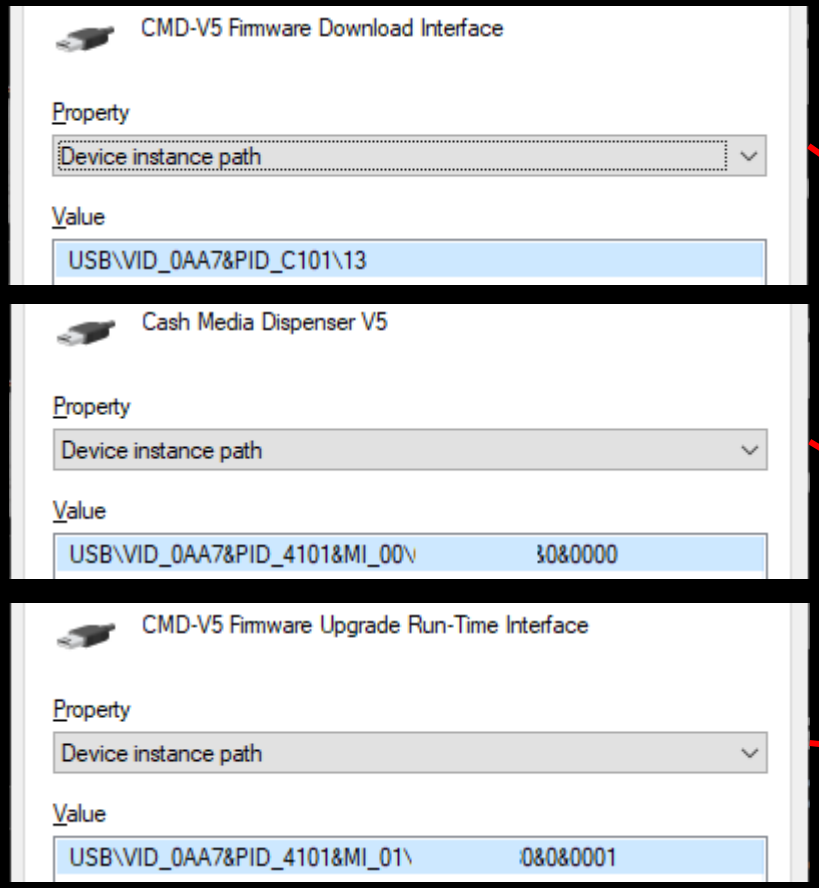


```
00000120  40 5C 04 16 01 00 00 00 01 00 00 00 0F 9B 5A 74  @\............Zt
00000130  F7 F6 01 00 24 4D 4F 44 24 20 31 34 31 31 32 38  ....$MOD$ 141128
00000140  20 31 30 30 32 20 43 44 35 5F 41 54 4D 2E 42 54   1002 CD5_ATM.BT
00000150  52 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  R...............
00000160  13 00 00 00 12 00 00 00 BE E3 CC 32 72 5F F6 1C  .........2r_..
00000170  90 D2 7A 2D 9F 49 38 04 CE 2E 0D 0E C8 91 AB 2E  ..z-.I8.........
00000180  27 BE D1 34 82 21 63 04 2C 55 74 06 03 AD 1A 11  '..4.!c.,Ut.....
00000190  F1 8E 01 05 01 FB 0C 16 79 25 3E 1F FC 18 30 2B  ........y%>...0+
000001A0  AA 11 99 04 3D 27 09 3F 39 C6 AA 04 0A 00 00 00  ....='.?9.......
000001B0  F0 52 C1 0F 02 3A CB 28 4F 89 E3 35 28 7C E3 39  .R...:.(O..5(|.9
000001C0  67 F1 BD 30 72 EA 38 2D 5D 51 1D 38 CB 89 75 10  g..0r.8-]Q.8..u.
000001D0  D5 38 96 14 59 B3 F9 06 C1 70 5C 2A 2D 43 88 17  .8..Y....p\*-C..
000001E0  D8 69 46 28 0C EE DC 25 43 7D 2C 3F AB DD 2F 33  .iF(...%C},?../3
000001F0  E7 60 B7 0C FF 4C F2 13 A8 5C 9D 1B D9 2B 04 32  .`...L...\...+.2
00000200  B3 C7 07 11 CB 00 5A 16 7D 06 80 10 CE 91 3A 09  ......Z.}.....:.
00000210  48 C9 62 30 DD D0 99 29 3F 10 BE 06 E9 31 70 26  H.b0...)?....1p&
00000220  E8 D4 7B 05 73 E9 59 3D E7 F5 66 38 93 24 6C 21  ..{.s.Y=..f8.$l!
00000230  7B 96 F0 16 0E 00 00 00 00 00 00 00 00 00 00 00  {...............
00000240  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000250  00 00 00 00 00 00 00 00 00 00 00 00 88 77 66 55  .............wfU
00000260  13 00 00 00 12 00 00 00 13 33 C3 2C E3 6C 51 00  .........3.,.lQ.
00000270  82 9A 09 03 A5 93 E8 28 78 BB CC 0F 7B 76 DB 2E  .......(x...{v..
00000280  8B 07 E1 0D 07 7F EB 3B 4A E8 47 04 70 7A 9A 1B  .......;J.G.pz..
00000290  4C 75 33 35 CF EB 4C 20 68 BA F7 11 B8 A9 1C 3E  Lu35..L h......>
000002A0  74 A0 84 16 4F BA E2 10 4D F8 75 28 0B 00 00 00  t...O...M.u(....
000002B0  9E 2A FE 31 B0 A2 FC 38 8E 54 67 22 0A 58 2D 10  .*.1...8.Tg".X-.
```

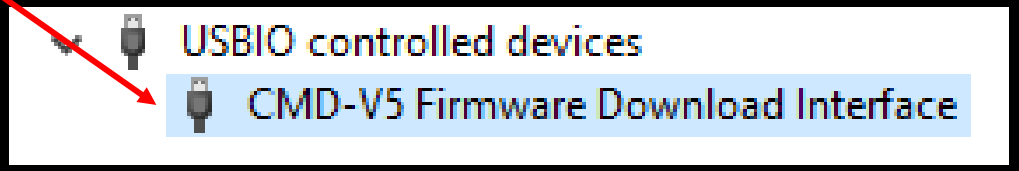
- 30-bit tokens count (*int length + 1*)
- 30-bit tokens count (*int length*)
- *sign* = RSA(e=7, SHA1(data[0x360:]))
- *modulus* = RSA.key.N

# Firmware analysis (CMDv5)

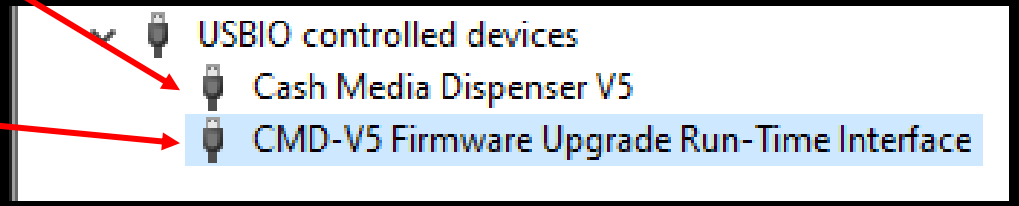
## Firmware uploading (DFU)




- Uses special DFU device:
  - DFU_PID = PID | 0x8000
  - bInterfaceClass = 0xFF
  - bInterfaceSubClass = 1



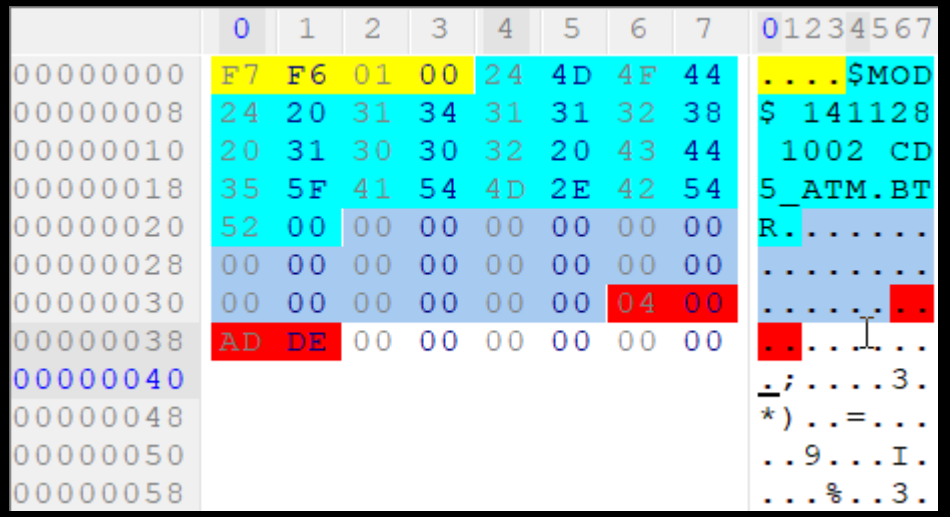

DFU-mode state




Normal state

# Firmware analysis (CMDv5)

## Firmware encryption tricks



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 01234567 |
|---|---|---|---|---|---|---|---|---|---|
| 00000000 | F7 | F6 | 01 | 00 | 24 | 4D | 4F | 44 | ....$MOD |
| 00000008 | 24 | 20 | 31 | 34 | 31 | 31 | 32 | 38 | $ 141128 |
| 00000010 | 20 | 31 | 30 | 30 | 32 | 20 | 43 | 44 | 1002 CD |
| 00000018 | 35 | 5F | 41 | 54 | 4D | 2E | 42 | 54 | 5_ATM.BT |
| 00000020 | 52 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | R..... |
| 00000028 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | ..... |
| 00000030 | 00 | 00 | 00 | 00 | 00 | 00 | 04 | 00 | ..... |
| 00000038 | AD | DE | 00 | 00 | 00 | 00 | 00 | 00 | ..... |
| 00000040 | | | | | | | | | .;....3. |
| 00000048 | | | | | | | | | *)..=... |
| 00000050 | | | | | | | | | ..9...I. |
| 00000058 | | | | | | | | | ...%..3. |

Firmware header

- **Unpacked FW size**
- **Firmware part name**
- **5 header-dwords**
- **0xDEAD0000 | (KEY0_OFFSET / 8)**

Bruteforce it!

Old keys checking code

```
if ( v16 >> 16 << 16 == 0xDEAD0000 )
{
  // pointing to old key0 (or to 0x00 filled area)
  xx1 = (int *)&old_keys0_block[8 * (unsigned __int8)v16];
  // pointing to old key1 (or to 0x00 filled area)
  xx2 = (int *)&old_keys1_block[16
                                * (((frm_buf[header_start + 0x16] << 16) |
                                   (frm_buf[header_start + 0x17] << 24) |
                                   (frm_buf[header_start + 0x15] << 8) |
                                   frm_buf[header_start + 0x14]) ^ *header_buf) & 0xFFF];
}
```

# Firmware analysis (CMDv5)

## KEY0 and KEY1:

- Hardcoded! (base offset: 0x64000000)
- Ability to use OLD or ZEROed keys!

```
// unxoring dword to check against 0xDEAD signature
v16 = ((frm_buf[header_start + 0x16] << 16) |
       (frm_buf[header_start + 0x17] << 24) |
       (frm_buf[header_start + 0x15] << 8) |
       frm_buf[header_start + 0x14]) ^ v1->header_5_dwords[0];
if ( v16 >> 16 << 16 == 0xDEAD0000 )
{
  // pointing to old key0 (or to 0x00 filled area)
  xx1 = (int *)&old_keys0_block[8 * (unsigned __int8)v16];
  // pointing to old key1 (or to 0x00 filled area)
  xx2 = (int *)&old_keys1_block[16
                              * (((frm_buf[header_start + 0x16] << 16) |
                                 (frm_buf[header_start + 0x17] << 24) |
                                 (frm_buf[header_start + 0x15] << 8) |
                                 frm_buf[header_start + 0x14]) ^ *header_buf) & 0xFFF];
}
// Forming two "data"-dwords
// which will be used as XTEA encrypted data
dest2_xx[0] = *xx1;
dest2_xx[1] = xx1[1];
```

# Firmware analysis (summary)

What we know:

1. Self-signed firmware (*public key is in the same binary!*)
2. APLib packed sequential blocks
3. Modified XTEA encryption algorithm (*different DELTA*)
4. XTEA encryption keys can be bypassed (VULN IS HERE!)
5. DFU protocol (*uploading firmware into a dispenser*)

# USB Communications (steps)

1. Basekey initialization
2. New session keys generation
3. Session counters synchronizing

# USB Communications (Basekey init)

To generate a new Basekey you need:

1. ROOT-certificate
2. Intermediate CA-certificate
3. Terminal Encryption certificate (issued by CA)
4. Terminal Authentication certificate (issued by CA)

We don’t have any of them… :(

(*and don’t need them*)

# USB Communications (session key)

How to generate a new session key (PC):

1. BK = Read the Basekey from the Keystorage (its key in TPM)
2. SESSION_KEY_XXX = SHA1(BK) + session_counter + direction

How to generate a new session key (Firmware):

1. SESSION_KEY_XXX = SmartCard(session_counter + direction)

SmartCard also checks for the same
counter usage + makes its increment

We have four directions:
PC_FW_OUT, PC_FW_IN, FW_PC_OUT, FW_PC_IN

# USB Communications (session sync)

To synchronize session counters you need:

1. ChannelID (server=2, client=1)
2. Basekey length
3. Basekey Check Value (KCV) (first 3 bytes of SHA1(Basekey)
4. Session counters for USB client/server IN/OUT

Response has the same parameters
so we can sync session counters

Basekey can be read from
the Keystorage file too

# Abusing session counters (DoS)

Steps to reproduce:

1. `session_counter = 0xFFFFFFFF`
2. `SESSION_KEY_XXX = SmartCard(session_counter + direction)`

SmartCard generates a new key,
but no new key can be generated after!

# USB comms analysis (summary)

## What we know:

1. TPM usage (*awesome!*)
2. Keystorage usage (*awesome!*)
3. Four encryption keys directions (*awesome!*)
4. SmartCard usage (*awesome!*)
5. SmartCard “*feature*”
   (*can disable a whole ATM, but won’t allow to take the money!*)

# USB Communications (withdrawal)

## Steps to perform a withdrawal:

1. Patch FW to skip asking SmartCard for a session key
   (use some dummy array)
2. Patch Java code to use the same dummy array as the key
3. Patch Java code to skip checks
   for cashIn and cashOut configs
4. Sync session counters (PC = SmarCard)
5. Write a new cassettes config to the dispenser's EEPROM
6. Call prepareCashOut()
7. Call cashOut(cassetteNum=3, banknotesNum=5)
8. Call shutter.open()
9. Take the money!
10. Close the shutter

# Vulnerabilities disclosure timeline

1. Q3 2018 – vendor has been informed about vulnerabilities
2. Q4 2018 – official PoC tests were performed, vulnerabilities have been proven
3. Q4 2018 – CVE IDs were registered
4. Q1 2021 – vendor informed us that vulnerabilities were fixed in 2019
5. Q3 2021 – <*Russian Mitre*> IDs:
   - BDU:2021-04967
   - BDU:2021-04968

# Thank you

Contacts:
vkononovich@ptsecurity.com