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

Windows-based application → XFS Layer

XFS Layer → XFS Service providers

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- Is obsolete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
Mostly unencrypted

Some vendors try to issue patches with communication encryption but they are limited by resources of old hardware.

In some cases, the protocol is ASCII-based, human-readable and looks like: “DGTM-01-02\n” that is an abbreviation of DispenserGimmeTheMoney from 1st 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

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 =)

Vendor selection - NCR

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

So... NCR S1 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 if you are not a service-man?

2. Where can you get the dispenser’s main board if you don’t work in a bank?

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
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
Beginning...

Some of search results (WritePacket, ReadPacket):

<table>
<thead>
<tr>
<th>Address</th>
<th>Function</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFC6F880</td>
<td>_WritePacket_Q3_9Universal5RTUSB16ColdfireEndpointFUi</td>
<td>move.l $104C(a1),d0</td>
</tr>
<tr>
<td>FFC6F88A</td>
<td>_WritePacket_Q3_9Universal5RTUSB16ColdfireEndpointFUi</td>
<td>move.l d0,$104C(a1)</td>
</tr>
<tr>
<td>FFC6FC2A</td>
<td>_ReadPacket_Q3_9Universal5RTUSB16ColdfireEndpointFUi</td>
<td>move.l $104C(a5),d0</td>
</tr>
<tr>
<td>FFC6FC34</td>
<td>_ReadPacket_Q3_9Universal5RTUSB16ColdfireEndpointFUi</td>
<td>move.l d0,$104C(a5)</td>
</tr>
<tr>
<td>FFC70074</td>
<td>_WritePacket_Q3_9Universal5RTUSB23ColdfireControlEndpointFUi</td>
<td>move.l $104C(a4),d0</td>
</tr>
<tr>
<td>FFC7007E</td>
<td>_WritePacket_Q3_9Universal5RTUSB23ColdfireControlEndpointFUi</td>
<td>move.l d0,$104C(a4)</td>
</tr>
</tbody>
</table>

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

```c
loc_FFC3AC78: movea.1 arg_0(a6), a5
clrw 0(a5)
clrl 2(a5)
```

```c
loc_FFC3AC78: movea.1 arg_0(a6), a5
clrw Cygnus::CommandPrimitives::CommsBuffer(a5)
clrl Cygnus::CommandPrimitives::CommsBuffer.capacity(a5)
```

```c
loc_FFC3AC78: movea.1 arg_0(a6), a5
clrw Cygnus::CommandPrimitives::CommsBuffer::buffer(a5)
```

General execution scheme

USB Receive Thread  
(service commands distribution)

Service1  
(Ex.: *DispTranService*)

- Serv1.Class1  
  (Ex.: *0x01*)

- Controller1  
  (Ex.: *StackController*)

Service2  
(Ex.: *securityService*)

- Serv2.Class1  
  (Ex.: *0x01*)

- Controller2  
  (Ex.: *PresentBillsController*)

- Serv2.Class2  
  (Ex.: *0x02*)

...
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

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
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!

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

Hello, Bootloader!
S1 (S2) “Secure” Bootloader

- Zlib-compressed code is located in “.data” section
- No symbols
- Image base is \texttt{0x100000}
- Is not secure!
- One wrong step – the dispenser will be bricked!
- Without a correct NVRAM-dump before 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

Going into bootloader

Only the first block

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

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

Also we can patch “secureCommand” function to be able to send any command without encryption.
S1 (S2) “Secure” Dispenser

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

![Dispenser Screen](image_url)
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] real banknotes from the [0x01, 0x02, 0x03, 0x04] virtual cassettes

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 00 00 00 00 00 00 00 |................|
00000010:  14 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 |................|
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**

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.

Thank you for listening!

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

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