CANANALYZE
A PYTHON FRAMEWORK
SSTIC 2020

ERWAN LE-DISEZ & ETIENNE CHARRON / 2020
AGENDA

# CONTEXT
# FRAMEWORK
# DEMO
# NEXT
01
CONTEXT
ARCHITECTURE OF A CAR

- **ECU (Electronic Control Unit)**
  - BCM (Brake Control Module)
  - Telematics box
  - Dashboard
  - …

- **BUS**
  - CAN (Controller Area Network)
  - I2C (Inter-Integrated Circuit)
  - LIN (Local Interconnect Network)
  - …
Cybersecurity impacts

- Safety (preserve passenger life) [*Main concern*] ★
- Data privacy (RGPD)
- IT (Automobile knowledge)

Scenarios

- Compromise an ECU in the multimedia network
- Bypass the CGW to send malicious frames in the vehicle network
SECURITY CONCERNS

- Verify Debug services are closed (or correctly locked by a robustness authentication)
  - UDS services (*Unified Diagnostic Services ISO 14229-1*)
    - ReadMemoryByAddress
    - WriteMemoryByAddress
    - Transfer data

- Verify sensitives frames are correctly filtered by CGW (CAN firewall)

*How to verify this? ... CANanalyze ...*
**GLOBAL OVERVIEW**

- **UDS**
  - **SERVICE_ID**
  - **PARAMETER1**
  - **VERY LONG PARAMETER2**

- **ISOTP**
  - **Fragmentation**
    - **FRAG**
      - **SERVICE_ID**
      - **PARAMETER1**
    - **FRAG**
      - **VERY LONG PARAMETER2**
      - **PAD**

- **CAN**
  - **CANID**
  - **DLC**
  - **C**
  - **FRAG**
    - **SERVICE_ID**
  - **PARAMETER1**
  - **CRC**

**UDS** (ReadMemoryByAddress, WriteMemoryByAddress, DataTransfer)

**Fragmentation**

**Simple packet (CANid DATA)**
02
FRAMEWORK
WHY CREATING A NEW FRAMEWORK?

Need for a CAN Army Swiss Knife

- Existing internal code base
- Programming language accessible to everyone, very simple API
- Support several hardware dongles (KOMODO, CANUSB)
- Support the use of several interfaces at the same time
- Specific features to validate / instrument CAN Gateways (virtual ECU / GW)
# EXISTING FRAMEWORKS

<table>
<thead>
<tr>
<th>Activity (GIT)</th>
<th>Udsoncan</th>
<th>CANTools</th>
<th>UDSim</th>
<th>CANanalyze</th>
</tr>
</thead>
<tbody>
<tr>
<td>★★★</td>
<td>★</td>
<td>★★★</td>
<td></td>
<td>Too recent</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Language</th>
<th>Python</th>
<th>Python</th>
<th>C/C++</th>
<th>Python</th>
</tr>
</thead>
</table>

| API simplicity   | ★★★      | ★        | ★★★   | ★★★★★     |

| Documentation    | ★★★      | ★★★      | ★★★   | ★★★★★     |

| CAN / ISOTP / UDS | ★★★      | ★★★      | ★★★   | ★★★        |

| ECU Simulator    | ✔️        | ✔️        | ✔️    | ✔️         |

| Script probing (CANid, UDS) | ✔️ | ✔️ | ✔️ | ✔️ |

| Hardware compatibility | ★★★ | ★★★ | ★★★ | ★★★★ |

---

Erwan LE-DISEZ & Etienne CHARRON / 2020
**Calibration**

JSON format defines routing + filtering per interface / CANID

```json
"dlc": {
  "ext": {
    "0x20": [{ "payload": "0x0000000000000000", "mask": "0xF0F0000000000000" },
      { "payload": "0x0040000000000000", "mask": "0xF0F0000000000000" }],
    "0x21": [{ "payload": "0x0000000000000000", "mask": "0xF0F0000000000000" },
      { "payload": "0x0040000000000000", "mask": "0xF0F0000000000000" }]
  },
  "v2": {
    "0x20": [{ "payload": "0x0000000000000000", "mask": "0xF0F0000000000000" }],
    "0x21": [{ "payload": "0x0000000000000000", "mask": "0xF0F0000000000000" }]
  }
}
```

**Interface mapping**

Specific mapping depending on the interfaces

```json
"interfaces": {
  "v1": { "channel": "vcan0", "bustype": "socketcan", "bitrate": 500000 },
  "v2": { "channel": "vcan3", "bustype": "socketcan", "bitrate": 500000 }
}
```

**Virtual Gateway**

Socket CAN Gateway: calibration.json + mapping.json

```
$ python3 scripts/gw_virtual_socketcan.py calibration.json mapping.json

Add virtual CAN interface vcan3 [physical=v1 virtual=vcan3]
Add virtual CAN interface vcan3 [physical=v2 virtual=vcan0]
Add virtual CAN interface vcan3 [physical=ext virtual=vcan1]
Add virtual CAN interface vcan2 [physical=dlc virtual=vcan2]

... READ
R: dlc [0x406 - 0xb'd20a38059b300e']
R: v1 [0x53f - 0xb'ae2f8f45d9e1']
R: dlc [0x200 - 0xb'df72']
R: v1 [0x7aa - 0xb'c5be5f348af39461']
R: dlc [0x405 - 0xb'67c68e0f3e093806']
R: v1 [0x7df - 0xb'6f33ee49fb21a96a']
R: v1 [0x202 - 0xb'12312333']
R: CAN ID matches = 0x020
F: v1 -> v2 [0x020 - 0xb'd212333']
W: v2 [0x020 - b'12312333']
R: v1 [0x021 - 0xb'aaaaaaa']
R: CAN ID matches = 0x021
F: v1 -> v2 [0x021 - 0xb'aaaaaaa']
W: v2 [0x021 - b'aaaaaaa']

... FORWARD
```

**Send messages to virtual GW:**

```
$ cangen vcan0
$ cansend vcan0 123#DEADBEEF

... WRITE
```
**Interface mapping**

Specific mapping depending on the interfaces

```
"interfaces": {
  "vl": { "channel": "vcan1", "bustype": "socketcan", 
    "bitrate": 500000},
  "ext": { "channel": "A", "bustype": "komodo", "port_nr": 1, 
    "bitrate": 500000},
  "dlc": { "channel": "B", "bustype": "komodo", "port_nr": 0, 
    "bitrate": 500000},
}
```

**Calibration**

Calibration depending on the hardware

Calibration only required to validate the routing and filtering configuration

**Validation script**

- Listen simultaneously on all interfaces and generate traffic depending on the tests
- Discover CANID authorized on interfaces (UDS DiagSessionControl)
- Check authorized CANID and payloads from calibration
**Goal:** Discover CANid offering UDS services (needed to get the debug services list)

```
$ python scripts/id_uds.py
km_init_channel: Acquired features: 38
km_init_channel: Bitrate set to 5000000
km_init_channel: Timeout set to 1 second(s)
UDS service detected (canid_send=0x7CA, canid_receive=0x7DA)
```
**Goal:** list UDS services exposed by the ECU (and verify that some UDS debug services are disabled)

```
$ python scripts/nmap.py
km_init_channel: Acquired features: 38
km_init_channel: Bitrate set to 5000000
km_init_channel: Timeout set to 1 second(s)
Scan.services discovered 10 Diagnostic Session Control
Scan.services discovered 11 ECU Reset
Scan.services discovered 14 Clear Diagnostic Session Information
Scan.services discovered 19 Read DTC Information
Scan.services discovered 22 Read Data By Identifier
Scan.services discovered 27 Security Access
Scan.services discovered 2e Write Data By Identifier
Scan.services discovered 31 Routine Control
Scan.services discovered 3e Tester Present
```
CAN abstraction interface
- Strong python-can adhesion: message format, socket CAN support (and more)
- Komodo support (single and dual interfaces)

ISOTP and advanced UDS interfaces

Context management
- Manage simultaneously multiple interfaces (CAN id filters, timeouts...)
- Per-context cache (with filtering capabilities)

```python
cxt = context.create_ctx (channel = 'A',
                         bustype = BusType.KOMODO,
                         port_nr = 0,
                         bitrate = 500000)

vcan.sniff (ctx, max=20)
vcan.write (ctx, can.Message(
            data = [0xD0, 0x32, 0x00, 0x09]), can_id = 0x166)
```
CANANALYZE

DEMO SETUP

- **4 virtual CAN interfaces:**
  - vcan0 (MULTIMEDIA) : exposed services
  - vcan1 (SAFETY) : sensitive ECU
  - vcan2 (ADAS) : optional driving aids
  - vcan3 (DIAG) : ODB II diagnostic

- **Sample calibration: ALLOW**
  - SAFETY => * : ALL CAN ID
  - ADAS => MULTIMEDIA : CANID 0x01 / ACK 0x02
  - DIAG => SAFETY : CANID 0x0a / ACK 0x0b
  - DIAG => ADAS : CANID 0x0d / ACK 0x0e

CANID routing
No payload filtering
04

EVOLUTION
FUTURE EVOLUTIONS

- Probing UDS routines
- Support more hardware dongle
- Support CANFD
- Automatize some tests on Security Access
- …
02’
COMMUNICATION WITH ECU
WHAT IS A CAN REQUEST?

- **CAN**
  - ISO 11898-2 (2003): CAN « high-speed » (until 1Mbits/s),

"Daisy-chain" structure with twisted-pair CAN High / CAN Low
UDS SERVICES

- **Services**
  - 0x10 / DiagnosticSession
  - 0x11 / EcuReset
  - 0x27 / SecurityAccess
  - 0x23 / ReadMemoryByAddress

- **Error Code**
  - 0x10 / generalReject
  - 0x11 / serviceNotSupported
  - 0x12 / subFunctionNotSupported
  - 0x35 / invalidKey
  - 0x33 / securityAccessDenied
HOW SEND DATA BIGGER THAN 8 BYTES?

**ISOTP**

- **0 = Single Frame**
  
  \[0x02, 0x10, 0x02, 0xFF, 0xFF, 0xFF, 0xFF\]

- **1 = First Frame**
  
  \[0x1X, 0xXX, 0xDD, 0xDD, 0xDD, 0xDD\]

- **2 = Consecutive Frame**
  
  \[0x21, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD\]
  \[0x22, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD\]
  \[0x23, 0xDD, 0xDD, 0xDD, 0xDD, 0xDD\]

- **3 = Flow Control Frame**
  
  \[0x30, 0xXX, 0xYY, 0x00, 0x00, 0x00\]

**Example**

- Send the following message from ECU A to ECU B

  \[0102030405060708090A0B0C0D0E0F101112\]

<table>
<thead>
<tr>
<th>CANid</th>
<th>C</th>
<th>DLC</th>
<th>DATA</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEB</td>
<td>X</td>
<td>8</td>
<td>10</td>
<td>12 01 02 03 04 05 06 XXXX...</td>
</tr>
<tr>
<td>DEB</td>
<td>X</td>
<td>8</td>
<td>20</td>
<td>07 08 09 0A 0B 0C 0D XXXX...</td>
</tr>
<tr>
<td>DEB</td>
<td>X</td>
<td>8</td>
<td>21</td>
<td>0E 0F 10 11 12 00 00 XXXX...</td>
</tr>
</tbody>
</table>
How send data bigger than 8 bytes?

Exchanged frames between the ECU A and ECU B:

- ECU A sends FF, ECU B sends FC, ECU A sends CF, ECU B sends CF.
CAN INTERFACE

- Hardware

<table>
<thead>
<tr>
<th>CAN interface</th>
<th>BeagleBone Black + Tranceiver</th>
<th>BeagleBone Black + extended CAPE</th>
<th>CANUSB dongle</th>
<th>Komodo CAN DUO</th>
<th>VECTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>COST</td>
<td>★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>API</td>
<td>Native Linux socketcan</td>
<td>Native Linux socketcan</td>
<td>Windows Library Native Linux socketcan</td>
<td>Windows/Linux C library + python binding</td>
<td>Windows environment / proprietary scripting</td>
</tr>
</tbody>
</table>

CAN connector D-SUB9 / ODB II (termination resistor)

- Software
  - Limitation of character device model and drivers implementation
  - Linux SocketCAN (>= 2.6.25) based on network layer
  - Advanced features and abstraction for user space applications
  - SocketCAN user space utilities and tools (can-utils)