CANANALYZE IS A PYTHON FRAMEWORK TO INTERACT WITH NORMALIZED PROTOCOLS (CAN, ISOTP, UDS) USED IN AUTOMOTIVE INDUSTRY. THIS FRAMEWORK WAS DEVELOPED TO HELP CYBERSECURITY VALIDATIONS, RUN DYNAMIC ANALYSIS OR VERIFICATIONS ON COMPLEX ECU (DETECT DEBUG SERVICES, TEST ROUTING AND FIREWALL CALIBRATION, INTERACT SIMULTANEOUSLY WITH AUTOMOBILE EQUIPMENT ON MULTIPLE INTERFACES). THE FRAMEWORK PROVIDES AN EASY TO USE API TO IMPLEMENT CUSTOM TESTS, A COLLECTION OF SCRIPTS FOR COMMON OPERATIONS INCLUDING VALIDATIONS ON VIRTUAL OR PHYSICAL GATEWAYS.

AUTHORS: ETIENNE CHARRON, ERWAN LE DISEZ

DATE: 2020/01/22

01

Context

Vehicle connectivity creates new risks. The complexity of the car's electronic systems is continuously growing with new challenges (autonomous driving, connectivity…). Car manufacturers and OEMs work to face the risks by strengthening the security of on board and off board systems. Circumstances that could be a common security risk in traditional IT systems, becomes a safety issues in the automotive.

The different ECUs (Electronic Control Units) use generally CAN (Controller Area Network) bus to exchange data or to be programmed. This technology is commonly used in automotive industry due to its cost and its simplicity. Other protocols have been normalized on top of the CAN, like UDS (Unified Diagnostic Services, ISO 14229-1) which offers high level services.

CAN bus is the central communication channel for all operations on ECUs. Because ECUs could have design vulnerabilities, bugs in implementation and because reverse engineering can be a complex and time-consuming task. Cybersecurity validations focus on external interfaces exposure. For example:

- Deactivation of debug services commonly used during the development part to program, configure or diagnostic (such as `Routine Control`, `Read Memory By Address`, `Write Memory By Address` with UDS services);
- Filtering of sensitive messages, like re programming message, to prevent compromise spreading...

To face these security challenges, new tools and new methods of validation must be invented, for example with a framework that could create an environment to interact between physical and virtual ECUs. CANanalyze framework doesn't reinvent the wheel, it is based on widely used python modules, and provides additional functionalities not found in other frameworks (udsoncan, CANToolz, UDSim...). Very simple to use and modular, it helps us to interact simultaneously with several interfaces (with different hardware dongles) and to emulate the behavior of complex ECU.02

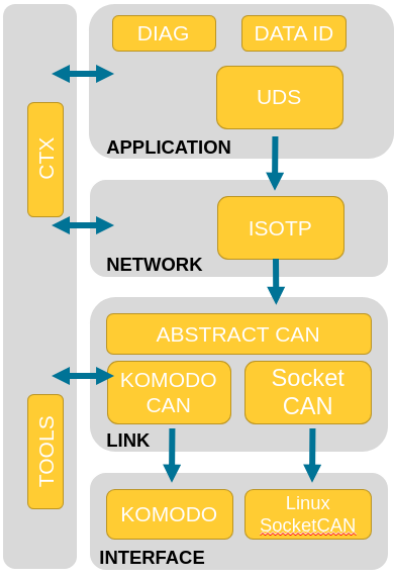
Framework

In this context, RENAULT and THALES have developed a python framework to help security validations. The framework can be used like a swiss army knife (toolbox), it has been designed to be easy to use with a collection of pre-defined scripts for common operations.

Implemented in python 3, it provides a simple API (`create ctx`, `read`, `write`) to send messages formatted in UDS, ISOTP or CAN protocol.

CAN abstraction interface uses `python-can` module as a data model for CAN messages. The CAN abstraction interface provides access to a wide choice of CAN interfaces (dongles and hardware modules), including socket CAN.

Additionally, the framework provides context management layer to manage simultaneously multiple interfaces with specific capabilities.



Framework architecture

Based on the `SocketCAN` support, the framework helps to implement virtual ECUs (virtual CAN interfaces) to simulate interactions between different components.

The following part describes some uses cases.

03

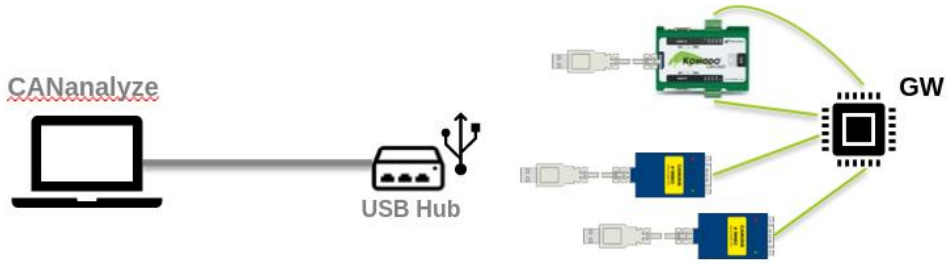
CAN interface modules

CANanalyze hides the low-level device-specific interfaces to controller area network adapters. It shares with `python-can` the same interface dependent modules and provide an abstraction for new ones.

Mainly used and validated interface modules are:

- CANUSB dongle using CAN overs Serial interface on Linux (slcan);
- Komodo CAN DUO with vendor software API on Linux.;
- Linux SocketCAN API.

These modules are not costly and allow anyone to interact easily with ECU composed of multiple interfaces (in theory no limit, tested with 8).



Sample usage with 4 CAN interfaces (2 CANUSB, 2 Komodo CAN DUO)

API

The framework exposes an API to easily write messages in different automotive protocols (CAN, ISOTP, UDS).

```
ctx = context.create_ctx (channel = 'A',  bustype = BusType.KOMODO,
                          port_nr = 0, bitrate = 500000, timeout = None)
acan.sniff (ctx, max=20)
acan.write (ctx, can.Message(data = [0xD0, 0x32, 0x00, 0x09]),
                             can_id = 0x166)
uds.write(ctx, [0x10, 0x02])
```

Script sending the CAN message 0xD0320009

Service Detection

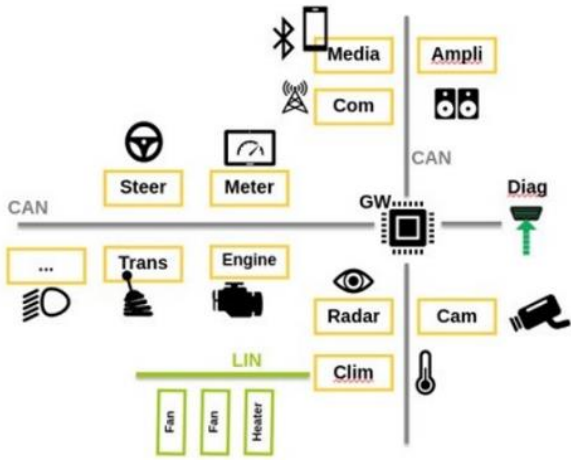
Some scripts have been developed with the framework, to detect the UDS services exposed by the ECU. The following screenshot shows the result of the scanner with an ECU.

```
$ 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
```

Script detecting UDS services on an ECU

Some UDS services can have a security impact by altering ECU integrity like `Write Memory By Address`, `Data Transfer`.

Gateways are automotive firewalls that isolate different CAN buses and filter CAN messages to prevent compromise spread. The figure on the right shows an architecture sample with a gateway. During a security validation, it is pertinent to check if the gateway is correctly calibrated.



The framework provides a calibration file format (JSON based) to define routing and filtering per interface and CANID.

To experiment and test a calibration file, a virtual gateway script is provided with the framework. This script emulates a gateway with as many interfaces as necessary and apply the behavior defined in the calibration file.

Virtual gateway script maps the virtual interfaces and dump CAN messages:

```
$ python3 scripts/gw_virtual_socketcan.py calibration_sample.json gw_mapping.json

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

READ: dlc [0x406 - 0xb'd20a38059b300e']
READ: v1 [0x53f - 0xb'ae2f8f45d9e1']
READ: v1 [0x020 - 0xb'12312333']
  READ: CAN ID matches = 0x020
    FORWARD: v1 -> v2 [0x020 - 0xb'12312333']
WRITE: v2 [0x020 - b'12312333']
READ: v1 [0x021 - 0xb'aaaaaaaa']
  READ: CAN ID matches = 0x021
    FORWARD: v1 -> v2 [0x021 - 0xb'aaaaaaaa']
WRITE: v2 [0x021 - b'aaaaaaaa']
```

Script emulating a gateway having 4 interfaces (v1, v2, ext and dlc)

And you can play with virtual interfaces with command line tools for example:

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

Script generating the interface vcan0 and sending the CAN message 0xDEADBEEF

The framework also provides a script to validate physical gateways. The script takes as input the definition of the interfaces (hardware modules, dongles...) and a calibration file. The script listens simultaneously on all interfaces and generate traffic depending on the tests. It discovers CANID authorized on interfaces and check authorized CANID and payloads depending on the calibration.