Structured Approaches to Automotive Cybersecurity Testing

IEEE SA - Standards for Trustworthy Autonomous Vehicles - Nurturing the Era of e2e Mobility as a Service (MaaS)

Stefan Marksteiner

Standards & Regulations

Today‘s connected vehicles are **insecure** from a cybersecurity perspective. There is **no system to** comprehensively and automatically **test the cybersecurity** of vehicles and their systems and subsystems. This topic is, however, becoming both so **important and complex** that such a system will be **heavily needed** – as a product as well as service.

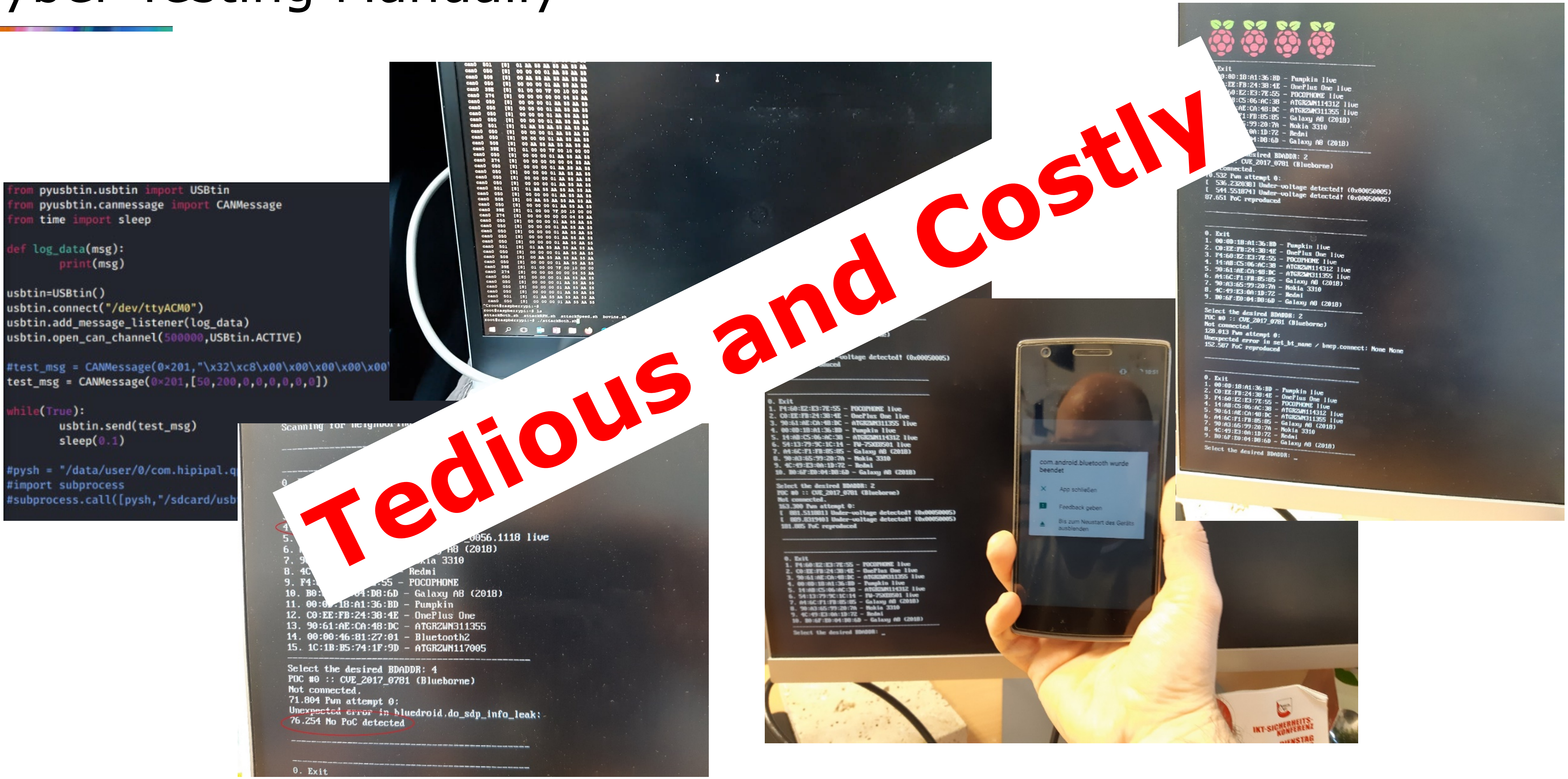
This is aggravated by standards' (ISO/SAE 21434) and regulators' (UNECE) requirements.

The Need for Industrialized Automotive Cybersecurity Testing

- UNECE
  - Regulation ECE/TRANS/WP.29/2020/79
  - Mandates cybersecurity and cybersecurity management
  - Requires testing of measures
  - Adopted in EU, Japan and Korea
  - Effective in EU for new types 2022 and for all new vehicles 2024
- ISO/SAE (DIS) 21434
  - Cyber security management system for automotive systems
  - Risk-based approach
  - Also demands testing, however, does not specify details
  - To be supplemented for testing by ISO/WD PAS 5112

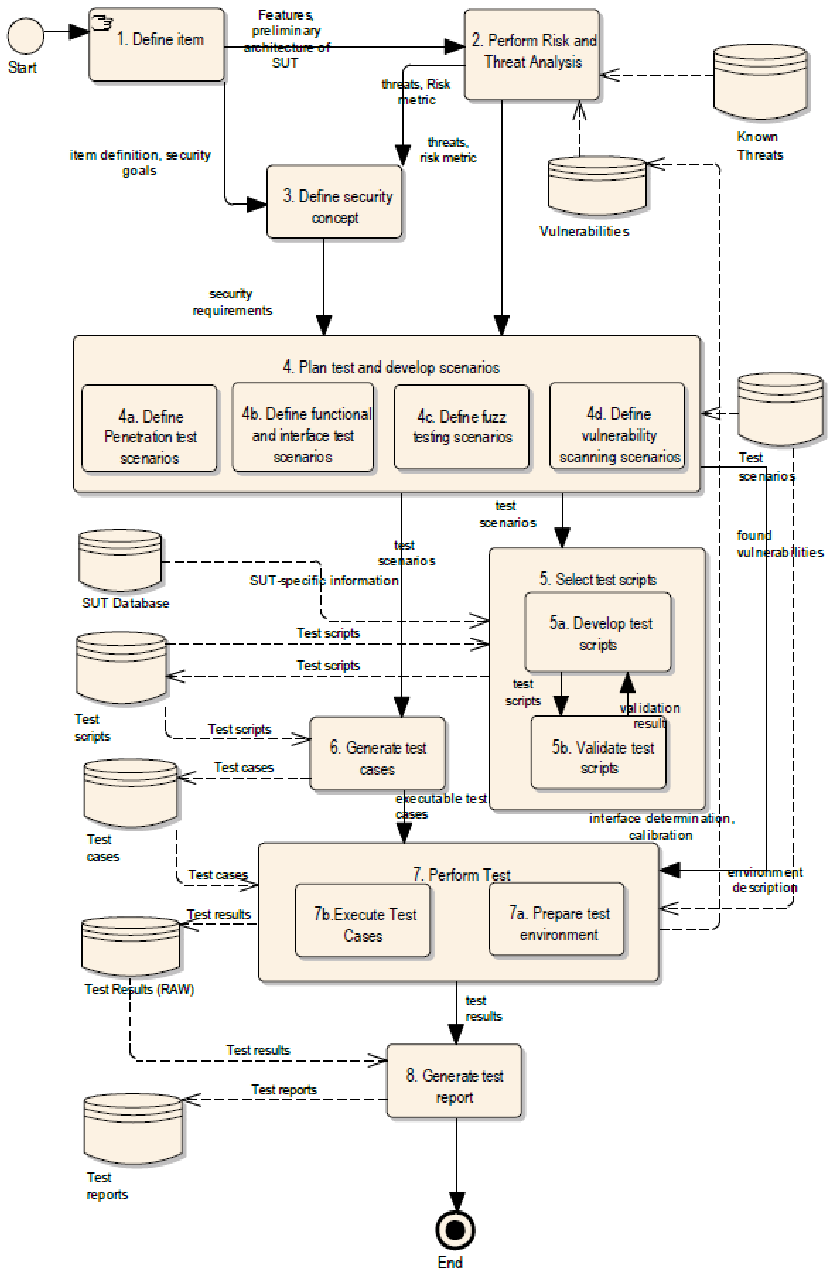
=> Need for automated testing over the whole life cycle

Cyber Testing Manually



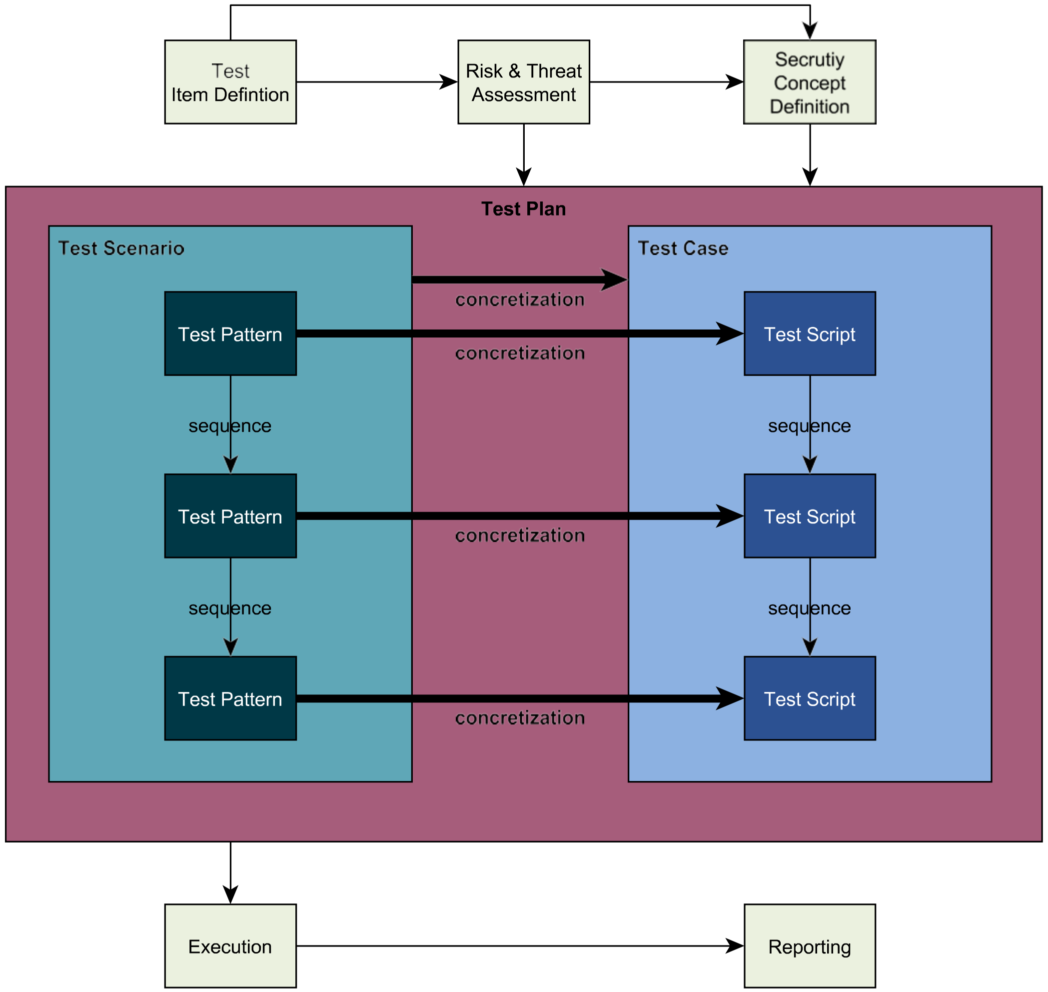
Automotive Cybersecurity Testing Process

- Systematic testing approach
- Targets towards automating testing
- Eight activities
  1. Define Item
  2. Perform Risk and Threat Analysis
  3. Security Concept Definition (mainly including the test targets)
  4. Plan Test and Develop Scenarios
     a. Define Penetration Test Scenarios
     b. Define Functional and Interface Test Scenarios
     c. Define Fuzz Testing Scenarios
     d. Define Vulnerability Scanning Scenarios
  5. Select Test Scripts
     a. Develop Test Scripts
     b. Validate Test Scripts
  6. Generate Test Cases
  7. Perform Test
     a. Prepare Test Environment
     b. Execute Test Cases
  8. Generate Test Reports.



Test Planning - Abstracting Test Patterns

- The main part of the process is defining test scenarios and generating test cases
- The relation between test scenarios and test cases are consists of abstraction and concretization
- The purpose is portability through generalization



Test-preparative Actions

- Define Item
  - Defines the test item (as needed for testing)
  - Item boundaries (context, environment, interfaces)
  - Functional description
  - Item model (or *candidate* black box testing)
- Perform Risk and Threat Analysis
  - E.g. TARA
  - Test priorization and non-testing
- Security Concept Definition
  - Test targets (building blocks from requirements)

Test Planning

- Create a realistic scenario of a cybersecurity attack
  - Penetration testing
  - Functional & interface testing
  - Fuzz testing
  - Vulnerability scanning
- Consists of abstract test building blocks
  - No SuT-specific information
  - Principal steps to perform an actual attack

```
ID <2> BT_Connect=TRUE
ID <4> MEASUREMENT(SPD, PRETEST)= 0
</PRECONDITIONS>
<ATTACK>
ID <1> Traget Vulns:=ACTION SCAN_IF_VULN  (Bluetooth, MA
ID <2> Shell:=ACTION EXPLOIT_BT (Target_Vulns, GetShell)
ID <3> RootShell:= ACTION OPEN_ADB_SHELL(ADB_KEY, S
ID <4> Result:=ACTION RUN_ATTACK_TOOL(RootShell, Can
</ATTACK>

<POSTCONDITIONS>
ID <2> BT_Connect=FALSE
ID <3> RootShell!=NULL
ID <4> Result=Success
ID <4> MEASUREMENT(SPD, INTEST)=200
ID <4> MEASUREMENT(SPD, POSTTEST)=0
```

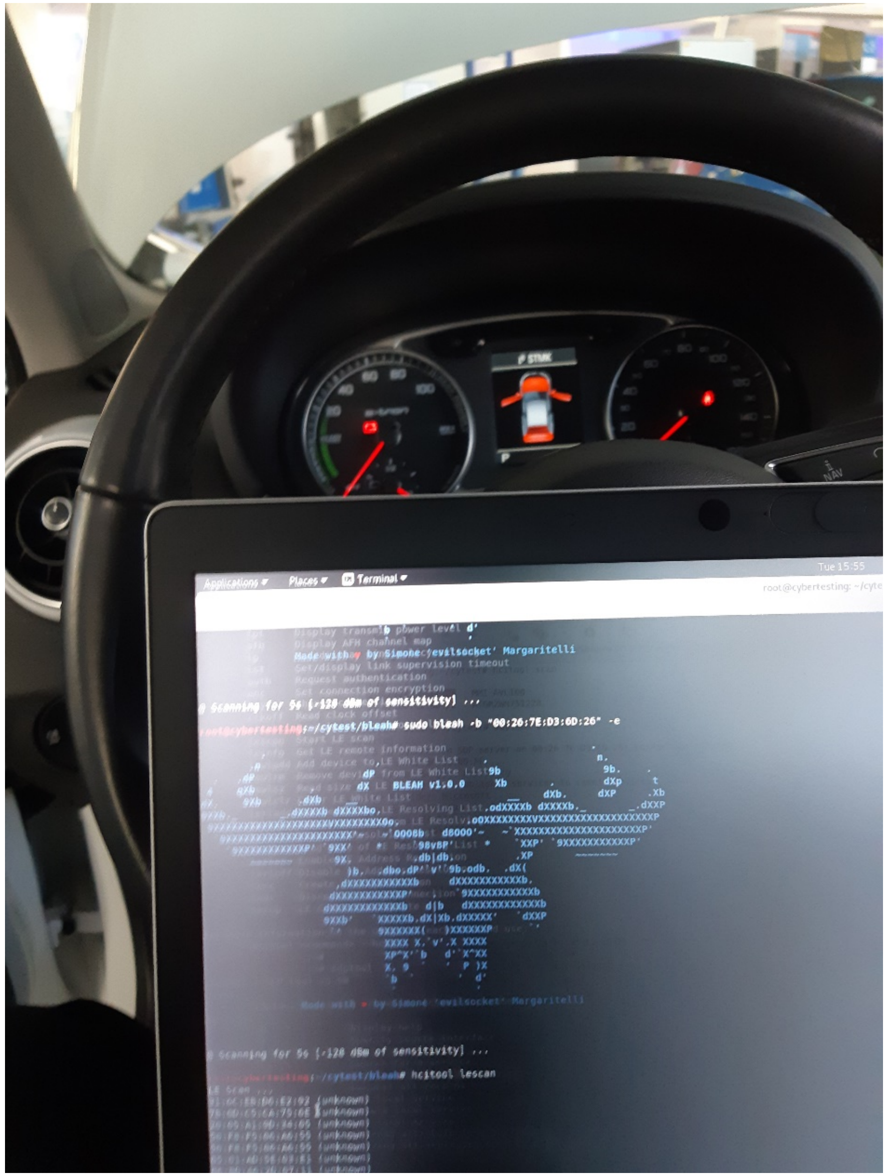
Script Selection and Test Case Generation

- Script Selection
  - Development of actual test scripts
  - Concrete, executable versions of attack patterns specific for distinct SuTs
- Test Case Generation
  - Attributes a known attack script/vulnerability to a step in the test scenario
  - Turns scenarios in executable test cases

```
from pyusbtin.usbtin import USBtin
from pyusbtin.canmessage import CANMessage
from time import sleep

def log_data(msg):
        print(msg)

usbtin=USBtin()
usbtin.connect("/dev/ttyACM0")
usbtin.add_message_listener(log_data)
usbtin.open_can_channel(500000,USBtin.ACTIVE)

#test_msg = CANMessage([redacted])
test_msg = CANMessage([redacted])

while(True):
        usbtin.send(test_msg)
        sleep(0.1)

#pysh = "/data/user/0/com.hipipal.qpy3/files/bin/qpython3-android5.sh"
#import subprocess
#subprocess.call([pysh,"/sdcard/usbtintest.py"])
```

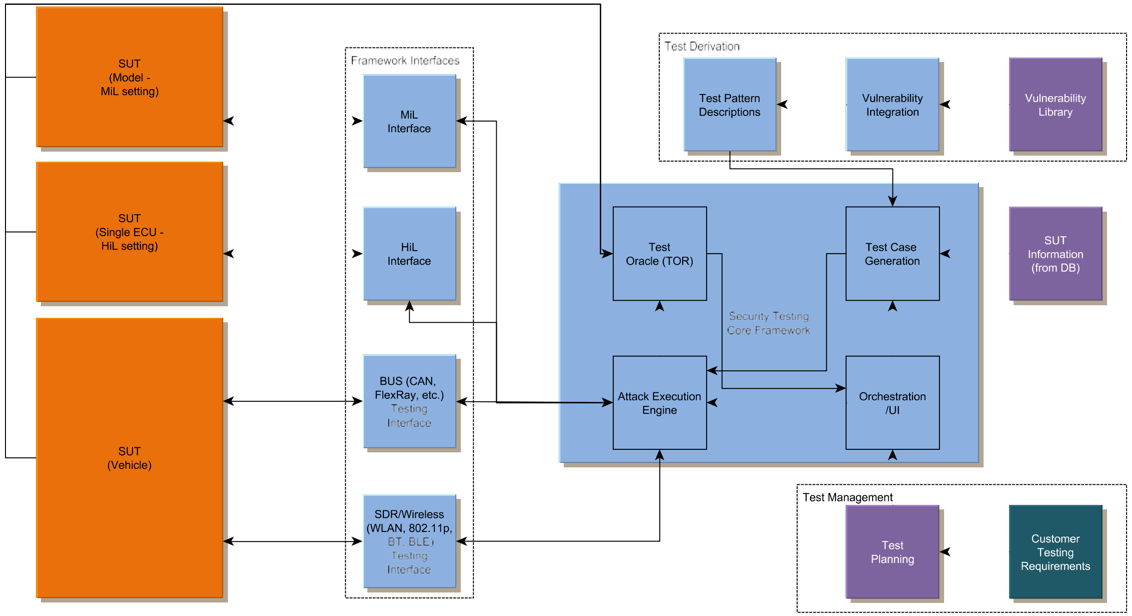
Test Execution

- Perform Test
  - Prepare Test Environment (commissioning, cleaning procedure)
  - Execute Test Cases
- Generate Test Reports



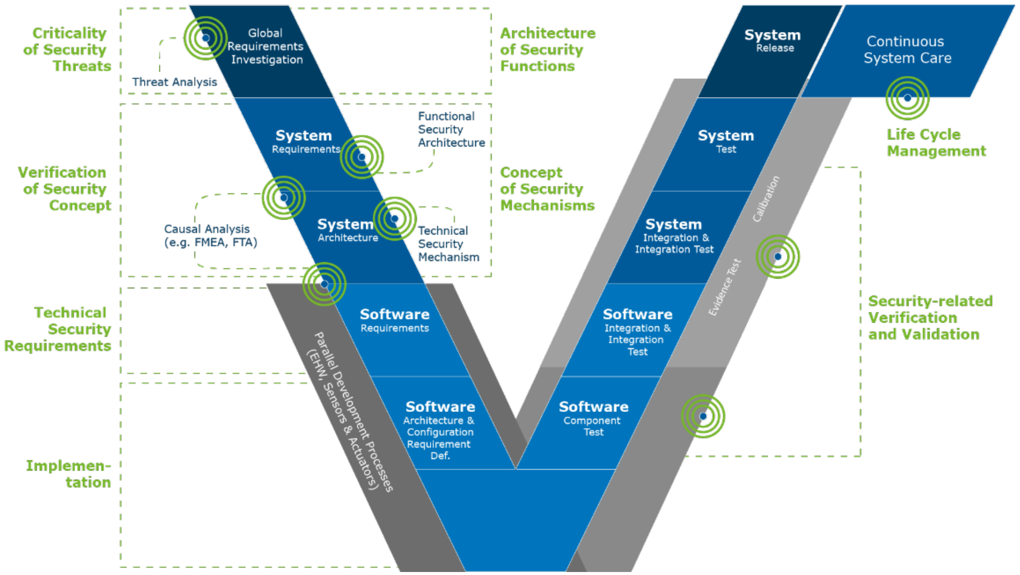
Concept Automotive Testing Framework

- A Framework that facilitates automated execution of the automotive cybersecurity testing process
- May consist of a core framework, test derivation, test management and interfaces
- Core FW with orchestration, test case generation, execution and test assessment
- Interfaces should be versatile for different types of SUTs to allow for different life cycle stages



Security Testing throughout the Whole Life Cycle

- Apart from traditional testing stages (right side of the V model), interfaces for (partly or fully) simulated are introduced:
  - Model-in-the-loop (MiL)
  - Software-in-the-loop (SiL)
  - Hardware-in-the-loop (HiL)
- The “tail” of the V model
  - Vulnerability management feeds test cases for incidents that emerge after the completion of the design
  - Software updates (OTA) could also be simulated first and real system-tested later to allow for full-life cycle testing



Conclusion

The process tries to address this and make automotive security testing:

- Automatable
- Comparable
- Efficient

Thank you for your attention!

IEEE SA STANDARDS ASSOCIATION

Thanks!

Stefan Marksteiner[1]

[1] Senior Technology Scout Cyber Security, AVL List Gmbh, stefan.marksteiner@avl.com

This work was supported by the H2020-ECSEL programme of the European Commission; grant no. 783119, SECREDAS project.