PRISSMA Project
Plateforme de Recherche et d'Investissement pour la Sûreté
et la Sécurité de la Mobilité Autonome
04/2021 - 04/2024

[L5.6] SECURITY TESTS REPORT OF AI BASED AUTONOMOUS SYSTEMES

Main authors: Lorenzo FABBRO, Romain GICQUAUD, Sammy HADDAD, Pierre LE MAISTRE, Romanos ZACHARAS.

Keywords: Vulnerability tests, Security tests, Vehicle, C-ITS-S, Evaluation

Abstract : This document presents the results of the vulnerability tests performed on a vehicle, a roadside unit, and on-board C-ITS equipment. These test targets have been provided to OPPIDA by UTAC. This deliverable describes the test targets and the different test environments, as well as a synthesis of the results. Due to sensitivity and confidentiality, we do not present full details of the tests and their results. We also present a list of tests that were initially meant to be performed but could not be conducted for various resource and technical reasons.

Résumé. Ce document présente les résultats des tests de vulnérabilité effectués sur un véhicule, une unité en bord de route et des équipements embarqués C-ITS. Ces cibles de test ont été fournies à OPPIDA par UTAC. Ce livrable décrit ces cibles et les différents environnements de test, ainsi qu'une synthèse des résultats. En raison de la sensibilité et de la confidentialité, nous ne présentons pas tous les détails des tests et de leurs résultats. Nous présentons également une liste des tests qui étaient initialement prévus mais qui n'ont pas pu être réalisés pour diverses raisons de ressources et techniques.

1 INTRODUCTION

The tests presented in this document were conducted as part of the PRISSMA project. The aim of the PRISSMA project is to define a methodology for evaluating the security of AI-based autonomous transportation systems. Among the various tasks and objectives of the project, WP5 defined a methodology for validating the cybersecurity of these systems. This methodology identified the need to validate the security of the system's critical components, including the AI-based autonomous vehicle and the Cooperative Intelligent Transport Systems Stations (C-ITS-S) providing secure communication services.
We aimed to test both kinds of elements (vehicle and C-ITS-S). However, since the project did not have autonomous vehicles available for cybersecurity tests due to resource constraints—the demonstrators were already allocated to numerous other test activities—tests were performed on a non-autonomous car.
UTAC provided Oppida with a more standard road vehicle featuring a comprehensive set of interfaces and technologies representative of both autonomous and non-autonomous vehicles. Vehicular technologies have similarities and common grounds for both autonomous and non-autonomous vehicles, including common standards, components, and architectures such as:

- Multimedia system (Android system Wi-Fi, Bluetooth, USB, etc.).
- IVN (CAN Bus, OBD)
- GSM/4G providing the ability, for example, to update various vehicle components.
- Etc.

Therefore, we conducted a cybersecurity testing campaign on this vehicle to identify possible vulnerabilities that might be present. These tests serve as reference points to evaluate the potential impact of similar vulnerabilities on autonomous vehicles, given their numerous shared components and technologies.
Tests on other critical components, such as the C-ITS components (i.e., communicating equipment that can provide strategic data to AI decision-based driving systems), were also conducted. These tests were performed on two roadside units (RSU) and one onboard unit (OBU).
This deliverable does not present full test details and results due to sensitivity and confidentiality constraints. These details were provided to partners involved in the evaluation.
As for deliverable 5.7, the goal of these tests is not to qualify specific products for which the developers were not involved and no specific support was provided. Rather, the goal is to demonstrate the type and associated efficiency of the performed tests themselves, including their efficiency in terms of (i) capabilities to demonstrate specific security properties, and (ii) cost and difficulty to perform the tests.
This document first presents the test environment used in section 2, including the test target and all the tools used during the tests, followed by a summary of the tests conducted and in section 3. This section presents the different test plans for the different targets and identify which tests were actually performed.
An initial set of tests has been identified for C-ITS-S components, based on SCA IRT systemX[1] project and extended for PRISSMA. However, some technical issues related to black-box testing conditions prevented us from performing all those tests. One main limitation was the impossibility of configuring the security functions of the C-ITS communications. The equipment configuration did not include any ETSI 103 097 certificates. Despite Eviden (not a

[1] https://www.irt-systemx.fr/projets/sca/

member of PRISSMA) graciously loaning the PRISSMA partner full C-ITS PKI access for this test, we did not manage to configure it and activate the signing and encrypting functions in this black-box context.
Therefore, we present both the intended and actual tests in this section. Finally, we present an anonymized summary of our findings in Section 4 and conclude the document in Section 0.

2 TEST PLATFORMS

2.1 Vehicle

Figure 1 Picture of one Renault E-Tech

The audited vehicle is a Renault Scenic E-Tech from year 2024. It is a test prototype loaned by Renault to UTAC. This vehicle was made available to Oppida to conduct tests as part of the PRISSMA project.
For these tests, we did not receive direct support or documentation from the manufacturer.
The vehicle provided the following interfaces:
CAN Bus providing access to the vehicle's internal network via physical access points identified under the passenger seat or in the trunk.

- OBD providing diagnostic capabilities (for example, it can detect defective vehicle components).
- GSM/4G providing the ability, for example, to update various vehicle components.
- SOS emergency call system compliant with the e-call standard.
- Tire pressure sensors.
- Passive keyless entry system.

To perform tests on the vehicle, it is necessary to use appropriate tools. The technologies and interfaces that make up these systems are not all part of standard Information Technology (IT) elements, but also part of the automotive industry standards.
Therefore, it is necessary to use specific tools suited to them as well as standard IT security tests tools. In this specific case, the tests were carried out using the following software and hardware tools:

- Hardware
    - PC avec machine virtuelle Kali Linux
    - Hackrf + Portapack (Mayhem Firmware version V2.0.0)
    - PCAN-USB
    - Adaptateur OBD2
    - Raspberry pi zero (RaspbianOS v12 bookworm)
    - Flipper Zéro avec firmware Xtreme en version 0052 du 08.12.2023
    - Antenne WiFi Alpha AWUS036ACM
- Software

- Nmap (version 7.80)
- Bluetoothctl (version 5.64)
- Bluing (version 0.12.0)
- Bluespy (https://github.com/TarlogicSecurity/BlueSpy)
- BlueDucky (https://github.com/pentestfunctions/BlueDucky)
- CaringCaribou (version 0.5)
- Bettercap (version 2.32.0)
- ADB (version 1.0.41)
- Python (version 3.8)
- Metasploit (version 6.3.55)
- Wifipumpkin3 v1.1.7
- Can-utils (Version 2023.03-1)

2.2 C-ITS-S components

Tests have been performed on two roadside units (RSU) and one onboard unit (OBU):

- RSU Lacroix RailDin
- RSU Lacroix
- OBU YoGoKo

The equipment has been provided by UTAC, along with part of the documentation.
For Lacroix equipment the configuration used is:

- RSU UEV EU - Configuration parameters Ed.17
- API RSU and OBU Ed.19
- Road unit User manual Ed.22

For YoGoKo equipment:

- YoGoKo-ITSUI APK
- Update binaries
  - 4.0.0.0_4.6.2.0.bin
  - 4.6.2.0_4.7.0.0_eu.bin
  - 4.6.7.0_5.0.1.2_eu.bin
- Quick setup guide YBOX-VEHI-2203 V1.0 (equivalent of the evaluated version)

No further technical documentation has been provided for the tests as well as no technical support from the developers.

3 TESTS

In this section, we present the test plans initially defined for the different targets identified in Section 2. Section 3.1 outlines the various planned tests, all of which were performed on the vehicle, while Section 3.4 details the tests planned for the C-ITS components, whether they were performed or not. The different types of tests (performed or not) are identified using tables of different colors (blue for tests that were performed and grey for tests that were not). For each test we describe we present them in the form of a table including:

- The function undertests - either identifying the interface, software or more specific function under test)
- A reference
- Authors - when identified (using evaluators trigram – first letters of the evaluator name and surname)
- Objective
- Initial conditions – preconditions necessary to the proper execution of the test
- Tools – tools used or required for the particular test.
- Tests steps – describing the different step performed during the test
- Expected result – presenting the expected behaviour to be observed for the test to validate its objectives

We do not present the actual detailed results, since the goal of the study is not to identify vulnerabilities or validate a specific implementation of a developer who is not even involved in the project. The main objective here is to present an overview of real test feedback: required equipment, limitations, cost and time, difficulties encountered, problems identified in the targets, tests conducted, etc. This analysis is provided in the following Section 4, based on the test details presented here.

3.1 Vehicle

3.2 WIFI

Functions: Wifi	**Reference: Test_Wifi_01**	**Author: PLM/LFA**
	Test purpose: Testing Wifi security protocol configuration and passwords	
Initial conditions:		
Access to the car Wifi		
Tools used:		
Test PC		
Test steps:		
Audit of the Wifi security configuration: • Cryptographic algorithms used • Identification of password currently used • Password policy enforcement		
Expected results:		
The observed secure communication protocol used are conform to state-of-the-art recommendations**,** no weak default passwords are used and weak password are refused.		

Functions: Wifi	Reference: Test_Wifi_02	Author: PLM/LFA
	Test purpose: Identification and validation of open ports.	
Initial conditions:		
Access to the car Wifi and credentials		
Tools used:		
Test PC with nmap		
Test steps:		
Scans performed: • With a connexion of the car android to an access point provided by the test PC • With a connexion of the test PC to the car Wifi access point		
Expected results:		
Only necessary open ports are identified**.**		

Functions: Wifi	Reference: Test_Wifi_03	Author: PLM/LFA
	Test purpose: Validation of the RTSP protocol configuration.	
Initial conditions:		
Access to the car Wifi and credentials		
Tools used:		
Test PC with python3.8		
Test steps:		
• RSTP Server scanning • Password bruteforce		
Expected results:		
Open RSTP port are not weak to brute force.		

Functions: Wifi	Reference: Test_Wifi_04	Author: PLM/LFA
	Test purpose: Perform social engineering attack using rogue access point.	
Initial conditions:		
Access to the car Wifi		
Tools used:		
Test PC with Wifipumpkin v1.1.7 WiFi Alpha AWUS039ACM Antenna		
Test steps:		

• Using Wifipumpkin to present a rogue Wifi connection interface to the car passengers vie the car android
Expected results:
The car android does not connect to the rogue AP or present a warning to the user about danger to connect to untrusted APs (suggesting not to connect, use VPN, .etc.).

3.3 Bluetooth

Functions:	**Reference: Test_bluetooth_01**	**Author: PLM/LFA**
Bluetooth	**Test purpose:** Service Discovery Protocol (SDP) parameters scanning and validation	
Initial conditions:		
Access to the car Bluetooth		
Tools used:		
Test PC with Bluetoothctl v5.64 and Bluing v0.12.0		
Test steps:		
• Target identification • Scanning of the target • Validation of the gathered information		
Expected results:		
All identified services are not subject to known vulnerabilities.		

Functions:	**Reference: Test_bluetooth_02**	**Author: PLM/LFA**
Bluetooth	**Test purpose:** Collect of vehicle information using the Bluetooth interface	
Initial conditions:		
Access to the car Bluetooth		
Tools used:		
Test PC with Bettercap (v 2.32.0)		
Test steps:		
• Scanning with Bettercap		
Expected results:		
No sensitive information (car configuration, user private data, etc.) can be collected.		

Functions:	**Reference: Test_bluetooth_03**	**Author: PLM/LFA**
Bluetooth	**Test purpose:** Exploit of known Bluetooth vulnerability and execution of arbitrary commands using dedicated tools	

Initial conditions:
Access to the car Bluetooth
Tools used:
Flipper zero
Test steps:
• Pairing of the flipper • Execution of the automated exploit
Expected results:
Pairing is in fact mandatory to perform the exploit making it irrelevant.

Functions:	**Reference: Test_bluetooth_04**	**Author: PLM/LFA**
Bluetooth	**Test purpose:** Exploit of known Bluetooth vulnerability and eaves-dropping of Bluetooth communications (BlueSpy).	
Initial conditions:		
Access to the car Bluetooth		
Tools used:		
Test PC with ***https://github.com/TarlogicSecurity/BlueSpy***		
Test steps:		
• Target scan • Execution of the automated exploit		
Expected results:		
No communications between the car and equipment connected to its Bluetooth are intercepted.		

Functions:	**Reference: Test_bluetooth_05**	**Author: PLM/LFA**
Bluetooth	**Test purpose:** Exploit of known Bluetooth vulnerability and execution of arbitrary commands using dedicated tools (BlueDucky)	
Initial conditions:		
Access to the car Bluetooth		
Tools used:		
https://github.com/pentestfunctions/BlueDucky Raspberry pi zero w PC Linux		
Test steps:		
• Automated scan and exploit execution		
Expected results:		

Exploits fail and no unauthorized actions are performed.

3.4 OBD

Functions:	Reference: Test_OBD_01	Author: PLM/LFA
OBD	**Test purpose:** Eavesdropping of CAN messages via OBD2 connection	
Initial conditions:		
Physical access to OBD port		
Tools used:		
OBD2 adapter Linux test PC with Caring caribou version 0.5 (https://github.com/CaringCaribou/caringcaribou)		
Test steps:		
• Connecting the PC to the OBD2 port • Eavesdropping of CAN message using automated tool		
Expected results:		
No traffic is Eavesdropped.		

Functions:	Reference: Test_OBD_02	Author: PLM/LFA
OBD	**Test purpose:** UDS diagnostic scan for IVN mapping	
Initial conditions:		
Physical access to OBD port		
Tools used:		
OBD2 adapter Linux test PC with Caring caribou version 0.5 (https://github.com/CaringCaribou/caringcaribou)		
Test steps:		
• **Connection to OBD2** • **Test PC CAN network interface configuration** • **Rogue diagnostic request without authentication**		
Expected results:		
No CAN IDS are obtained.		

Functions:	Reference: Test_OBD_03	Author: PLM/LFA
OBD	**Test purpose:** Execution of rogue ECUs' reset commands	
Initial conditions:		

Physical access to OBD port Scan of ECUs
Tools used:
OBD2 adapter Linux test PC with Caring caribou version 0.5 (https://github.com/CaringCaribou/caringcaribou)
Test steps:
• Send RESET command to already identified ECU
Expected results:
No arming RESET can be executed.

3.5 Can Bus

Functions: CAN	Reference: Test_CAN_01	Author: PLM/LFA
	Test purpose: CAN Eavesdropping	
Initial conditions:		
Physical access to CAN bus wires		
Tools used:		
CAN-USB adapter Linux test PC with can-utils version 2020.11.0-1		
Test steps:		
• Identification of CAN wires • Physical plugin to CAN wires • Test PC Network interface setup • CAN eavesdropping		
Expected results:		
No sensitive traffic eavesdropped.		

Functions: CAN	Reference: Test_CAN_02	Author: PLM/LFA
	Test purpose: Replay of CAN traffic	
Initial conditions:		
Physical access to CAN bus wires and eavesdropping set-up		
Tools used:		
CAN-USB adapter Linux test PC with can-utils version 2020.11.0-1		

Test steps:
• Capture traffic • Extract potential sensitive execution commands • Replay commands
Expected results:
No armful commands (lights, engine, windshield weeper) are executed.

Functions: CAN	**Reference: Test_CAN_03**	**Author: PLM/LFA**
	Test purpose: Forging and sending of rogue traffic	
Initial conditions:		
Physical access to CAN bus wires and eavesdropping set-up		
Tools used:		
CAN-USB adapter Linux test PC with can-utils version 2020.11.0-1		
Test steps:		
• Reverse engineering of traffic • Forging of new commands • Sending of new commands		
Expected results:		
No armful commands (lights, engine, windshield weeper) are executed.		

3.6 OSINT

Functions: ALL	**Reference: Test_OSINT_01**	**Author: PLM/LFA**
	Test purpose: Identification of known attacks for the target or sensitive security application	
Initial conditions:		
Internet access		
Tools used:		
Dorks Google, Shodan.io, Censys, Bing		
Test steps:		
• Documentation and vulnerabilities look-up • Identification of potential vulnerabilities and exploit • Execution of potential exploits		
Expected results:		
No exploit identified in the state-of-the art.		

3.7 GSM/LTE

Functions: GSM/LTE	Reference: Test_GSM_01	Author: PLM/LFA
	Test purpose: Man in the middle attack	
Initial conditions:		
Access to LTE emissions		
Tools used:		
OpenBTS / OpenLTE BladeRF / USRP		
Test steps:		
• Antenna set-up • **Interception of GSM/LTE radio messages**		
Expected results:		
No exploit identified in the state-of-the art.		

3.8 Android

Functions: Android	Reference: Test_Android_01	Author: PLM/LFA
	Test purpose: Identification of open debug port	
Initial conditions:		
Physical connection to the vehicle or Wifi access to the vehicle		
Tools used:		
Linux test PC with Adb (1.0.41), Nmap (v 7.94)		
Test steps:		
• Adb and nmap scan • Analysis of open ports		
Expected results:		
No debug port identified.		

Functions: Android	Reference: Test_Android_02	Author: PLM/LFA
	Test purpose: Execution of malware	
Initial conditions:		
Physical connection to the vehicle or Wifi access to the vehicle		
Tools used:		
Kali Linux test PC with Metasploit		

Test steps:
• Packing of a malware to be sent to the android • Sending and execution of the malware
Expected results:
No malware is executed.

Functions: Android	**Reference: Test_Android_03**	**Author: PLM/LFA**
	Test purpose: Identification of Android configuration	
Initial conditions:		
Physical connection to the vehicle or Wifi access to the vehicle		
Tools used:		
N/A		
Test steps:		
• Identification of Android version using user interface • CVE look-up for the obtained version		
Expected results:		
No CVE identified.		

3.9 Other radio media

Functions: TPMS	**Reference: Test_TPMS_01**	**Author: PLM/LFA**
	Test purpose: Eavesdropping of TPMS traffic	
Initial conditions:		
Access to vehicle radio emissions		
Tools used:		
Hackrf-One and Portapack		
Test steps:		
• Trame capture • Identification of potentially sensitive data in the traffic		
Expected results:		
No sensitive data identified.		

Functions: Start vehicle	**Reference: Test_Start_01**	**Author: PLM/LFA**
	Test purpose: Start vehicle using radio emissions	

Initial conditions:
Access to vehicle radio reception range
Tools used:
Hackrf-One Test PC linux with Gnuradio-companion
Test steps:
• Capture start radio trame • Replay start trames
Expected results:
Replay does not allow unauthorized user to start the vehicle.

3.10 C-ITS-S

3.11 Secure association

Functions: **Certificate management**	**Reference**: Test_SA_1 **Author**: RGI **Test purpose:** Ensure that the equipment cannot accept an invalid certificate as a parameter (expired, malformed, with an unsupported protocol, etc.).

Initial conditions:
Access to the OBU interface where certificates are loaded.
Tools used:
Burpsuite
Test steps:
1. Take a valid certificate and check that it is accepted by the application. 2. Changing a field contained in the signed part. 3. Check that the certificate is denied. 4. Repeat on all fields. 5. Send a self-signed certificate and verify rejection.
Expected results:
Only correctly signed certificates should be accepted.

Functions: **Certificate management**	**Reference**: Test_SA_2 **Author**: RGI **Test purpose:** Ensure that the equipment cannot accept an invalid certificate as a parameter (expired, malformed, with an unsupported protocol, etc.).

Initial conditions:
Access to the OBU interface where certificates are loaded.
Tools used:

Burpsuite

Test steps:

1. Take a valid certificate and check that it is accepted by the application.
2. Changing a field contained in the signed part.
3. Check that the certificate is denied.
4. Repeat on all fields.
5. Send a self-signed certificate and verify rejection.

Expected results:

Only correctly signed certificates should be accepted.

Functions:	**Reference**: Test_SA_3	**Author**: RGI
Certificate management	**Test purpose:** Verify the constraints imposed by EtsiTs103097.	

Initial conditions:

Access to the OBU interface where certificates are loaded.

Tools used:

Titan: https://forge.etsi.org/rep/ITS/ITS/tree/STF525

Test steps:

1. "_CertificateId_" is either of type "_name_" or "_none".
2. cracaId_ = 000000'H
3. crlSeries = 0'D
4. includes a "_validityPeriod".
5. at least one component "_appPermissions_" or "_certIssuePermissions_" shall be present and define signing permission.
6. the component "_encryptionKey_" of type "_PublicEncryptionKey_".
7. the digest of certificate has been calculated using proper hash algorithm (SHA-256 or SHA-384) after certificates canonicalization.
8. Check that end_validity is greater than "start_validity".
9. Check that all AIDs containing in the in the "its_aid_list" are unique.
10. The subject_name variable-length vector has a maximum length of 32 bytes.

Expected results:

All constraints must be implemented and compliant.

Functions:	**Reference**: Test_SA_4	**Author**: RGI
Certificate management	**Test purpose:** Ensure that the equipment cannot accept an invalid certificate as a parameter (expired, malformed, with an unsupported protocol, etc.).	

Initial conditions:

Access to the OBU interface where certificates are loaded.

Tools used:

Titan: https://forge.etsi.org/rep/ITS/ITS/tree/STF525

Test steps:

1. Fuzz one field.
2. Try to have the station validate the certificate.
3. Retry.

Expected results:

Only valid certificates should be accepted by the station.

Functions:	Reference: Test_SA_5	Author: RGI
Certificate management	**Test purpose:** Ensure that, within a certificate chain, the correct certificates are accepted and the others are rejected.	

Initial conditions:

Access to the OBU interface where certificates are loaded.

Tools used:

None

Test steps:

1. Create a certificate chain with invalid certificates
2. Verify which ones are rejected and which ones are accepted

Expected results:

Only valid certificates should be accepted.

3.12 Message protection

Functions:	Reference: Test_MP_1	Author: RGI
Message protection	**Test purpose:** Ensure that the equipment accept only correctly signed messages.	

Initial conditions:

Access to the OBU interface where certificates are loaded and to the EBR.

Tools used:

GNURadio with (https://github.com/bastibl/gr-ieee802-11)
Titan (https://forge.etsi.org/rep/ITS/ITS/tree/STF525)

Test steps:

1. Intercept signed OBU radio messages.
2. Modify the message without changing the signature (or without the signature).
3. Check whether the message is accepted by the ToE.

Expected results:

Only correctly signed certificates should be accepted.

Functions: Certificate management	Reference: Test_MP_2	Author: RGI
	Test purpose: Ensure that changing the security header does not skip signature verification.	

Initial conditions:
Access to the OBU interface where certificates are loaded.
Tools used:
GNURadio with (https://github.com/bastibl/gr-ieee802-11) *Titan (https://forge.etsi.org/rep/ITS/ITS/tree/STF525)*
Test steps:
1. Intercept signed OBU radio messages. 2. Modify the security header (can be fuzzed). 3. Check whether the message is accepted by the ToE.
Expected results:
Only a message with a correct security header and enabling signature should be accepted.

3.13 Replay protection

Functions: V2X communications	Reference: Test_Replay_1	Author: RGI
	Test purpose: Verify that legitimate and unique packets or messages are accepted without issues and validate that replayed packets or messages are correctly detected and rejected in different message load conditions.	

Initial conditions:
Access OBU and EBR
Tools used:
GNURadio with (https://github.com/bastibl/gr-ieee802-11) *Wireshark* *USRP*
Test steps:
1. Capture a C-ITS packet (CAM, DENM, SPAT or MAP). 2. Replay immediately the unique non-replayed captured packet using SDR. 3. Verify if it is accepted.
Expected results:
The ToE must detect that the packet has already been sent and refuse it.

Functions: Software update	Reference: Test_Replay_2	Author: RGI
	Test purpose: Ensure the the timestamp contained in the message is verified by the ToE.	

Initial conditions:
Access OBU and EBR.
Tools used:
Gnuradio *Wireshark with (https://github.com/bastibl/gr-ieee802-11)* *USRP*
Test steps:
1. Capture a packet (CAM, DENM, SPAT, MAP). 2. Replay after delay the unique non-replayed captured packet using SDR. 3. Verify if it is accepted again.
Expected results:
The ToE must detect that the packet has already been played and refuse it.

3.14 Privacy

Functions:	**Reference**: Test_Privacy_1	**Author**: RGI
Software update	**Test purpose:** Verify that it is not possible to exhaust the AT pool of TOE.	

Initial conditions:
Access OBU and EBR.
Tools used:
Gnuradio, wireshark, lib gnuradio (à préciser), hackrf, usrp
Test steps:
1. Fuzz the TOE with messages to attempt modifying the AT. 2. If a message is accepted, change the AT as much as possible.
Expected results:
The TOE does not change from the AT permanently, but it has indeed a cooldown between every modification.

3.15 Plausibility Checks

The following tests were carried out using Lacroix equipment only. They are therefore not consistent, as they should be carried out from an OBU.

Functions:	**Reference: Test_MBD_1**	**Author: RZA**
OBU logic	**Test purpose:** Verify the consistency of the speed limit.	
Initial conditions:		
OBU access.		

Tools used:
Burpsuite
Test steps:
Case 1: Set a value over threshold for speed acceptance. Case 2: Set a speed data value not compatible with acceleration.
Expected results:
Receive an error message from the application for incompatible values.

Functions:	**Reference: Test_MBD_2**	**Author: RZA**
OBU logic	**Test purpose:** Verify the consistency of event position (eg not on the road, over water, etc).	
Initial conditions:		
OBU access.		
Tools used:		
Burpsuite		
Test steps:		
Case 1: Set a value incompatible with speed of heading. Case 2: Set a position not on the road. Case 3: Set a position that overlaps with other vehicles or incompatible solid objects. Case 4: Set a position too far from the receiver.		
Expected results:		
Receive an error message from the application for incompatible values.		

Functions:	**Reference: Test_MBD_3**	**Author: RZA**
OBU logic	**Test purpose:** Verify the consistency of road headings.	
Initial conditions:		
OBU access.		
Tools used:		
Burpsuite		
Test steps:		
Case 1: OBU heading direction not compatible with speed. Case 2: OBU settled speed not compatible with road heading.		
Expected results:		
Receive an error message from the application for incompatible values.		

Functions:	**Reference: Test_MBD_4**	**Author: RZA**

OBU logic	**Test purpose:** Verify the consistency of vehicle length and wide
Initial conditions:	
OBU access	
Tools used:	
Burpsuite	
Test steps:	
Case 1: Identify threshold for vehicle length and width.	
Expected results:	
Receive an error message from the application for incompatible values.	

Functions:	Reference: Test_MBD_5	Author: RZA
OBU logic	**Test purpose:** Verify the consistency of road curvatures.	
Initial conditions:		
OBU access.		
Tools used:		
Burpsuite		
Test steps:		
Case 1: Incompatible with speed or heading changes.		
Expected results:		
Receive an error message from the application for incompatible values.		

3.16 Access Control

Functions:	Reference: Test_AC_1	Author: RZA
Web access control	**Test purpose:** Verify the presence of an anti-brutforce system.	
Initial conditions:		
Access to the equipment's web connection interface.		
Tools used:		
Hydra v9.5		
Test steps:		
• The auditors run a bruteforce attack on the Web interface of the LaCroix equipment in order to test if the application would block any bruteforce attempt. The username “oppida” was chosen on purpose to make a numerous failed login attempts as it does not exist. The password wordlist used in this attack is rockyou.		

- The attack shows that even after a numerous amount of failed login attempts the bruteforce keeps going.
- The auditors also run a test attack with the given credentials in order to assure the bruteforce is executing correctly.

Expected results:

Being blocked after a certain amount of failed login tries.

Functions:	Reference: Test_AC_2	Author: RGI
Web access control	**Test purpose:** Verify the presence of session fixation.	

Initial conditions:

Authenticated access to the equipment's web interface.

Tools used:

Burp, Web browser Developer Tools Interface

Test steps:

1. Login to the application with an account.
2. Note the value of the session cookie.
3. Log in again and check that the now cookie is different from the first one.

Expected results:

The cookie changes between each connection.

Functions:	Reference: Test_AC_3	Author: RGI
Web access control	**Test purpose:** Verify that the session token generated is fairly random.	

Initial conditions:

Authenticated access to the equipment's web interface.

Tools used:

Burp Sequencer

Test steps:

1. Capture a connection request (login request).
2. Send the request to the sequencer and run the sequencer until at least 10,000 tokens are obtained.
3. Analyze the result.

Expected results:

Burp must notify us that the token alea is sufficiently large.

Functions:	Reference: Test_AC_4	Author: RGI

Web access control	**Test purpose:** Verify that the system has appropriately configured session timeout settings to prevent unauthorized access due to idle sessions.
Initial conditions:	
Authenticated access to the equipment's web interface.	
Tools used:	
Burp, Web browser Developer Tools Interface	
Test steps:	
1. Identify the storage method for user sessions. 2. Test session replay after using the logout method (i.e. logout button). 3. Test replaying a session after it has expired.	
Expected results:	
The website must return that the session has expired.	

Functions:	Reference: Test_AC_5	Author: RGI
Web access control	**Test purpose:** Evaluate how passwords are stored, ensuring they are properly hashed and salted.	
Initial conditions:		
Access to the TOE's web interface and command line with password storage documentation.		
Tools used:		
None		
Test steps:		
1. Check documentation to ensure that passwords are hashed with a robust hash function before being stored. 2. Check at the place where passwords are supposed to be stored, that they are indeed hashed with a robust algorithm and a random & different salt.		
Expected results:		
The documentation must reflect reality, and a robust hash function must be used for password storage (i.e. SHA-512, etc). Furthermore, hash salts must be different, or very rarely the same, between two hashes.		

Functions:	Reference: Test_AC_6	Author: RGI
Web access control	**Test purpose:** Test the security and resistance to account takeover attempts in the password reset functionality.	
Initial conditions:		

Authenticated access to the equipment's web interface.
Tools used:
Burp, Web browser Developer Tools Interface
Test steps:
1. Reset several times the password of an account with an e-mail address you own. 2. Compare the link used to perform the reset to check that it is a non-deterministic link. 3. Try to reuse an old link to try to reset a password a second time. 4. Check if it's possible to do a CSRF password reset with the link.
Expected results:
The link sent with each reset request must be different and non-deterministic. Using a link that has already been used, or that is not the last link sent by the application, must not allow the password to be reset.

Functions:	**Reference: Test_AC_7**	**Author: RGI**
Web access control	**Test purpose:** Confirm that failed login attempts are correctly logged, and the system can trigger alerts for suspicious activity.	
Initial conditions:		
Access to the equipment's web connection interface and to the log interface.		
Tools used:		
Hydra v9.5		
Test steps:		
1. Bruteforce with a single user and a password list. 2. Check that alert logs have been raised. 3. Bruteforce with a list of users and a single password. 4. Check that alert logs have been raised. 5. Bruteforce with a list of users and a list of passwords without using a password and a user twice. 6. Check that alert logs have been raised.		
Expected results:		
Steps 2, 4 and 6 should raise alert and generate logs notifying a bruteforce attempt.		

Functions:	**Reference**: Test_AC_8	**Author**: RGI
Web access control	**Test purpose:** Examine the system's vulnerability to create or modify system-level processes to execute malicious payloads.	
Initial conditions:		
Access OBU		

Tools used:
gdb *ptrace*
Test steps:
1. Find vulnerabilities enabling to upload and execute code (buffer overflows, poor access right management of uploaded files, command injection in user inputted data, etc.) 2. Try to load shellcode into process
Expected results:
Processes cannot be injected.

Functions:	**Reference: Test_AC_9**	**Author: RZA**
Web access control	**Test purpose:** Evaluate the system's security against attempts to break out of a container to access the underlying host.	
Initial conditions:		
Container access.		
Tools used:		
Docker		
Test steps:		
• Correctly configured docker containers will not allow commands like fdisk -l. However, on miss-configured docker where the flag "–privileged or –device=/dev/sda1" with caps is specified, it is possible to get the privileges to see the host drive. Trying to mount filesystem to attacker define folder. • If there is an access as root inside a container that has some folder from the host mounted and have escaped as a non-privileged user to the host with read access over the mounted folder, a bash suid file can be created inside the container.Executing this file from the host will lead to privesc. • In case the command docker exec can be executed as root using sudo, a CVE can lead to escalating privileges form a container abusing CVE-2019-5736. This exploit will overwrite /bin/sh binary of the host from a container, so anyone executing the docker may trigger the payload. docker run –rm -it -v /:/host ubuntu bash		
Expected results:		
The container is well-secured and it's impossible to break out.		

Functions:	**Reference**: Test_AC_10	**Author**: RZA
Web access control	**Test purpose:** Examine the manipulation of how the operating system locates programs or libraries, potentially including malicious payloads.	
Initial conditions:		

Access OBU.

Tools used:

Pspy (on linux)

Process Monitor (on Windows)

Test steps:

If the user has written permissions on any folder inside the PATH variable, there might be a possibility to hijack some libraries or binaries.

```
echo $PATH
```

Check for the version of installed packages and services. These could be exploited for privilege escalation.

```
dpkg -l
```

Look at what processes are being executed and check if any process has more privileges than it should. The use of pspy is recommended.

```
ps aux
```

Check if any scheduled job is vulnerable.

```
crontab -l
ls -la /etc/cron* /etc/at*
ls -la /etc/cron* /etc/at* /etc/anacrontab /var/spool/cron/crontabs/root 2>/dev/null | grep
-v “^#”
```

If a cron script is executed by root and ca be modified, this could lead to a very easy privilege escalation.

```
echo ‘cp /bin/sh /tmp/bash; chmod + s /tmp/bash’ > </PATH/MALICIOUS/CRON/SCRIPT>
##wait for the cronjob to be executed##
/tmp/bash -p
```

This command lists the current user’s sudo privileges.

```
$ sudo -l
User user123 may run the following commands on env123
   (ALL) SETENV: /opt/scripts/admin_tasks.sh
```

In this scenario user123 can run admin_tasks.sh with any environment variables they choose to set, due to (ALL) SETENV directive.
This could be vulnerable to PYTHONPATH hijacking in order to load an arbitrary library while executing the script as root.

```
Sudo PYTHONPATH=/dev/shm /opt/scripts/admin_tasks.sh
```

A malicious python module can be placed in /dev/shm that matches an import statement in admin_tasks.sh. So, when the script is run with elevated privileges via sudo, it will import the malicious python module, leading to a potential privilege escalation.
If the sudo permission is given to a single command without specifying the path.

```
User123 ALL=(root) less
```

```
export PATH=/tmp:$PATH
##Put malicious code in /tmp and name it “less”##
sudo less
```

Expected results:

Only administers should be able to rewrite libraries and files used in the system execution flow. The same goes for paths containing files and/or folders.

Functions:	Reference: Test_AC_11	Author: RGI
Web access control	**Test purpose:** Check for the absence of SUID/SGID on vulnerable binaries.	

Initial conditions:

TOE access via command interface.

Tools used:

LinPEAS

GTFObins

Test steps:

1. Run LinPEAS or a command to list all SUID/SGID on the machine.
2. Analyze the output and see if it's possible to execute commands or elevate privileges with the listed applications (with GTFObins).

Expected results:

None of the applications listed by LinPEAS, or a command listing SUID/SGID, should be vulnerable to command execution or privilege escalation.

Functions:	Reference: Test_AC_12	Author: RGI
Web access control	**Test purpose:** Examine the system for adversaries attempting to elevate privileges using sudo caching or by modifying the sudoers file.	

Initial conditions:

TOE access via command interface.

Tools used:

None

Test steps:

1. Check sudo version.
2. Check sudo user privileges (on files or directories).
3. Check whether it is possible to write to /etc/sudoers.

Expected results:

Two results are expected:

1. Sudo is not installed on the machine, so no privilege elevation is possible with it.
2. The sudo version is not vulnerable to a known attack, and there are no sudo user privileges or privileges that can be elevated. Finally, the /etc/sudoers file must not be modifiable by any user (other than root).

Functions:	Reference: Test_AC_13	Author: RZA
System access control	**Test purpose:** Access to a command line interface through physical interfaces	

Initial conditions:
Physical access to the LaCroix equipment.
Tools used:
Saleae, *Screen,* *RS232 reader*
Test steps:
Using Saleae, a logic analyzer tool, the auditors obtained a capture of the data transmitted serially from the DEBUG port on the LaCroix machines. Upon further analysis using the async serial analyzer which decodes serial data one bit at the time from the capture file the Baud Rate was determined to be of 8 .7us. The baud rate is defined as the number of signal units transmitted per second. Given this information, the bit rate can be calculated using the following formula: *Bit Rate = Baud Rate x the number of bits per baud* By calculating the correct Bit Rate, the output appears in a readable format.
Expected results:
Serial analysis of data transmitted serially from the DEBUG port. Expected a capture of the communication.

3.17 Initialization

Functions:	Reference: Test_Init_1	Author: RGI
Web access control	**Test purpose:** A specific log entry shall be returned.	

Initial conditions:
Access the TOE logging interface.
Tools used:
None

Test steps:
1. Clear or note the last logs. 2. Reboot the TOE. 3. Check the log interface for a new entry specifying TOE startup.
Expected results:
A new entry has been added to the log.

3.18 Trust elements update

Functions: Time	**Reference**: Test_TrustElUp_1 **Author**: RGI **Test purpose:** Verify if the system requires administrative privileges and logs the modification for changing the system clock through the operating system's settings and control panel.

Initial conditions:
Access OBU
Tools used:
None
Test steps:
1. Change the internal clock time without system privilege. 2. Change the internal clock time with system privilege.
Expected results:
The system only lets us make the changes if we have privileged right. However, the modifications are logged.

3.19 Timestamp

Functions: Time	**Reference**: Test_Time_1 **Author**: RGI **Test purpose:** Ensure that the TOE clock is synchronized with trusted external servers.

Initial conditions:
Access OBU
Tools used:
None
Test steps:
1. Test to manually change (as a non-privileged user) the synchronization to see if accepts the modification.

2. Test to manually change (as a privileged user) the synchronization to see if accepts the modification.

Expected results:

The system only lets us make the changes if we have privileged right. However, the modifications are logged.

Functions: Time	**Reference**: Test_Time_2	**Author**: RGI
	Test purpose: Verify if the system detects and rejects unauthorized modifications.	

Initial conditions:

Access OBU

Tools used:

None

Test steps:

1. Modify the configuration files bound to the parameters of internal clock management.
2. Check if the system detects or at least rejects the operation if it does not have administrator rights.

Expected results:

System detects modification of clock configuration files

Functions: Time	**Reference**: Test_Time_3	**Author**: RGI
	Test purpose: Verify if the system detects the discrepancy between the hardware clock and software clock.	

Initial conditions:

Access OBU

Tools used:

Screwdrivers

Test steps:

1. Disconnect the equipment from the network.
2. Put the equipment out of tension by taking off the clock out of the hardware (don't forget to remove the battery from the motherboard).
3. Restart the equipment.

Expected results:

On restart, the ToE should realize and notify that it has been completely disconnected from the network (i.e. it can no longer synchronize with remote servers) and that the power-down has probably affected its internal clock.

Functions:	**Reference**: Test_Time_4	**Author**: RGI

Software update	**Test purpose:** Verify if clock settings can be modified through BIOS menu.

Initial conditions:
Access OBU
Tools used:
None
Test steps:
Change the clock settings from the BIOS menu.
Expected results:
On reboot, the ToE should realize that the BIOS and the network are indicating two different times. It must then synchronize with the network.

3.20 Enrollment

Functions: PKI managment	**Reference**: Test_Enrol_1 **Author**: RZA **Test purpose:** Verify that the TOE send secured requests to the PKI when enrolling or requesting for new EC.

Initial conditions:
ToE system access.
Tools used:
Titan: https://forge.etsi.org/rep/ITS/ITS/tree/STF525
Test steps:
1. Modify the current EC information provided in the PKI request to verify that a new EC based on known attacker information cannot be sent to the TOE. 2. Add additional unsigned or unencrypted field to the request to make the PKI generate a new certificate based on those rogue elements known to the attacker. 3. Replace the data encryption key sent to the PKI and used to encrypt the answer containing the replied EC. 4. Eave drop the PKI request and try to extract the data encryption key to latter disclose EC sent to the TOE.
Expected results:
EC Exchanges are carried out securely, regardless of the fields or data sent to the ToE.

3.21 Authorization

Functions: PKI managment	**Reference**: Test_Auth_1 **Author**: RZA **Test purpose:** Verify that the TOE send secured requests to the PKI when requesting for new ATs.

Initial conditions:
ToE system access.
Tools used:
Titan: https://forge.etsi.org/rep/ITS/ITS/tree/STF525
Test steps:
1. Try to modify the current EC information provided in the PKI request to verify that AT based on attacker information cannot be sent to the TOE. 2. Add additional unsigned or unencrypted field to the request to make the PKI generate a new certificate based on those rogue elements. 3. Replace the data encryption key sent to the PKI and used to encrypt the answer containing the replied EC. 4. Eave drop the PKI request and try to extract the data encryption key.
Expected results:
EC Exchanges are carried out securely, regardless of the fields or data sent to the ToE.

3.22 Check operation

Functions:	**Reference: Test_CheckOp_1**	**Author: RZA**
Physical test	**Test purpose:** Escaping U-Boot	
Initial conditions:		
Physical access on the LaCroix equipment.		
Tools used:		
RS-232 reader, *Screen*		
Test steps:		
To read the UART protocol the auditors connected a RS-232 analyzer on the DEBUG port. Using the ‘screen’ application, facilitates an interactive session. Executing a screen command to reveal the machine booting through U-Boot. The process can be interrupted by pressing a random key on the keyboard. Upon interrupting the bootloader, the auditors were presented with the U-Boot menu. Using the ‘printenv’ command, recovered all environment variables necessary for a safe boot. By modifying a specific parameter in the boot process, a root shell can be activated while booting.		
Expected results:		
Get an interactive session from the bootloader.		

Functions:	Reference: Test_CheckOp_2	Author: RZA
Bootloader	**Test purpose:** Escaping U-Boot Ventana	
Initial conditions:		
Physical access to the Yogoko equipment.		
Tools used:		
RS-232, screen, reader/writer mSATA for Solid State Drive		
Test steps:		
To read the UART protocol connect a RS-232 analyzer on the DEBUG port. Use the 'screen' application. Execute a command to reveal the machine booting through U-Boot. The process can be interrupted by pressing a random key on the keyboard. Upon interrupting the bootloader, the U-Boot menu is accessible. Then modify bootloader's variables		
Expected results:		
Like Lacroix equipment the auditors followed the same process to escape the bootloader since both are using U-Boot.		

Functions:	Reference: Test_CheckOp_3	Author: RZA
Physical test	**Test purpose:** Escaping U-Boot Ventana	
Initial conditions:		
Physical access to the Yogoko equipment.		
Tools used:		
RS-232 reader, *Screen,* *reader/writer mSATA for Solid State Drive*		
Test steps:		
Read the Solid-State Drive using a mSATA reader. Mount theSolid-State Drive to a known point with read and write rights. Add an ssh key to gain root ssh access.		
Expected results:		
By modifying the filesystem, the equipment should not be usable anymore.		

3.23 Audit

Functions:	Reference: Test_Audit_1	Author: RGI
Web access control	**Test purpose:** A specific log entry shall be returned.	

Initial conditions:
Access the TOE logging interface.
Tools used:
None
Test steps:
1. Clear or note the last logs. 2. Reboot the TOE. 3. Check the log interface for a new entry specifying TOE startup.
Expected results:
A new entry has been added to the log.

Functions:	**Reference: Test_Audit_2**	**Author: RGI**
Web access control	**Test purpose:** Verify that the logs generated contain sufficient data to identify the type of log, the time the log was recorded and the user who generated it.	
Initial conditions:		
Access to the ToE and access to the logging documentation.		
Tools used:		
None		
Test steps:		
1. Study of logging documentation 2. Checking implementation		
Expected results:		
Logging must be sufficiently consistent to be able to trace important actions carried out on the system (who, when, where, what).		

3.24 Software update

Functions:	**Reference**: Test_SWU_1	**Author**: RGI
Software update	**Test purpose:** Ensure that it's not possible to downgrade the ToE.	
Initial conditions:		
Access OBU and software updates		
Tools used:		
None		
Test steps:		
1. Identify the software version used		

2. Take an earlier version than the current one
3. Try to downgrade it.

Expected results:

ToE must not accept the update.

Functions:	**Reference**: Test_SWU_2	**Author**: RGI
Software update	**Test purpose:** Verify that only authenticated and validated software images can be used by the TOE for updates.	

Initial conditions:

Access OBU and software update

Tools used:

None

Test steps:

1. Modify update packet (one bit is enough) and proceed to upload it.
2. Check if there is a verification of the signature when uploading the update.

Expected results:

4 TESTS RESULTS SUMMARY AND SYNTHESIS

4.1 Vehicle

4.2 Exhaustivity

The tests performed on the vehicle were not exhaustive as it is usually the case in cybersecurity. It is always difficult to do exhaustive tests since exhaustivity is both (i) time consuming and (ii) difficult to demonstrate (formal proof?). As presented in Section 3 some planned tests have not been performed due to either resource constraints or technical difficulties that can be overcome with additional efforts and more experience on the targeted technology: Test_GSM_01, Test_Start_01. Also with growing maturity, more tests could and should be specified to further reach the exhaustivity challenge.
One of the many pieces of feedback from this test campaign is that a vehicle is a large system of systems in which many components could deserve on their own the time spent on the entire vehicle. Such elements as the infotainment system, IVN gateway, and various radio communication interfaces could deserve weeks of evaluation as defined in our assurance framework.
One limitation of the study is the lack of tests on AI. However, even if AI systems are an additional critical part of the global system, we have been able to identify that they are just one among many components. Also, their security heavily relies on the overall security of their environment even more than their own weaknesses regarding known attacks. There are so many potential ways to take control of vehicle subsystems that could directly impact any AI decisions, that clearly the cybersecurity validation cannot focus on the sole AI components.

4.3 Cost and time

The tests were performed by two persons over three weeks. This might seem extensive for non-cybersecurity experts, but for evaluation laboratories used to performing high-assurance evaluations on much smaller IT targets, which typically represent 4 to 6 full months of tests, this is not excessive.
As proposed in deliverable 5.3, the most critical components of the car should have their own dedicated evaluation, each with similar dedicated resources to those involved here. It should be noted that with more effort on specific vehicle components, the global vehicle validation should be easier, as part of the tests performed here would be covered by the targeted component evaluation.

4.4 Limitations

No critical limitation was identified besides the previously mentioned issues of resources and lack of AI target.

4.5 C-ITS-S components

4.5.1. Exhaustivity

An extensive test plan was identified, but a very limited set of tests were performed (less than 20%) due to limitations identified in Section 4.5.3.

4.5.2. Cost and time

Similar resources to the car tests were involved. The targets were smaller (interfaces and complexity of the services offered), and this level of effort seemed to be acceptable to achieve the

appropriate assurance (high assurance as defined in deliverable 5.3). But the limitations identified in Section 4.5.3 make it difficult to assert that.

4.5.3. Limitations

One major limitation arose during our tests. The security features (signature and encryption of messages) could not be set up even though they actually were the main target of the tests.

4.6 Findings summary

In this section, we present the overall summary of all findings on all targets identified in Section 2. Again, the idea here is not to qualify the specific targets we had to test but to showcase the number and type of issues that can be identified with respect to the resources, test plan, and target types. Therefore, this summary only presents the overall number of vulnerabilities found and their severity.

Severity	Nbr of vulnerability identified
High	6
Medium	7
Low	2

Table 1 Vulnerability findings summary

CONCLUSION

The cybersecurity tests conducted as part of the PRISSMA project have highlighted several key aspects and challenges in evaluating the security of AI-based autonomous transportation systems.

Part of the tests required specific equipment and expertise, as they were not regular IT tests. These included connection to the IVN and C-ITS-S communication security protocols. However, many COTS interfaces were found to be vulnerable to regular IT attacks. The car presented all main communication means regularly provided to the car owner, such as Wi-Fi, Bluetooth, and LTE. One might assume that limited access to such services should be provided to STRA users. However, current trends in transport services seem to demonstrate that access to various kinds of information or infotainment services is provided to passengers. It wouldn't be surprising if this were also the case for ARTS. Although we don't recommend it, these tests clearly demonstrate that if those interfaces exist, they'll need a strong security focus and validation since they are vectors for many known attacks.

Several critical findings were identified. Very specific expertise was required to perform security tests. The collaboration between Oppida (cybersecurity experts) and UTAC (automotive experts) was essential to perform these tests. STRA clearly requires both to perform successful security assurance evaluations.

Although we did not test AI, we observed that the vehicular attack surface (autonomous or not) provides many regular IT vulnerabilities that need to be tested and removed from any ARTS systems. Not all the vulnerabilities found during these tests would be present on ARTS, since not all communication interfaces we tested might be available. However, it is very likely, based on the project demonstrators, that such communication interfaces (for remote supervision, software updates, V2X communications, users' infotainment services) will continue to be present or aim to be included in the future.

Additionally, specific care should be taken in reviewing security assumptions. In our tests, we identified vulnerabilities that seem to be covered by specific assumptions made by the developers. Such assumptions as physical access, user security awareness, absence of vulnerabilities in COTS, and radio signal propagation limitations should be carefully evaluated. In our study, we believe that some of these identified vulnerabilities might actually be exploitable by motivated attackers.