Ref. Ares(2021)7498824 - 05/12/2021



ICT-56-2020 "Next Generation Internet of Things"
Grant Agreement number: 957218

IntellioT

Deliverable D4.4
Trust mechanisms (first version)

Deliverable release date	30/11/2021
Authors	1. EURECOM: Jérôme Härri 2. AAU: Beatriz Soret, Lam Nguyen 3. TSI: Andreas Brokalakis, Vassilis Amourgianos, Babis Savvakos 4. SANL: Konstantinos Fysarakis, Ioannis Vezakis
Editor	Konstantinos Fysarakis (SANL)
Reviewer	Sumudu Samarakoon (UOULU), Nancy Irisarri Mendez (PHILIPS)
Approved by	PTC Members: (Vivek Kulkarni, Konstantinos Fysarakis, Sumudu Samarakoon, Beatriz Soret, Arne Bröring, Maren Lesche) PCC Members: (Vivek Kulkarni, Jérôme Härri, Beatriz Soret, Mehdi Bennis, Martijn Rooker, Sotiris Ioannidis, Anca Bucur, Georgios Spanoudakis, Simon Mayer, Filippo Leddi, Harshitha Chandregowda, Maren Lesche, Fragkiskos Parthenakis)
Status of the Document	Final
Version	1.0
Dissemination level	Public

Table of Contents

1 Introduction 4
2 Continuous Assurance and Certification 7
2.1 Overview 7
2.2 Internal Architecture 7
2.2.1 Cyber System Asset Loader 9
2.2.2 Vulnerability Analyzer 10
2.2.3 Dynamic Tester 11
2.2.4 EVEREST Monitoring 12
2.3 The IntellIoT Assurance Model 14
2.4 Model-driven Assessments 17
3 Authentication, Authorisation and Accounting 21
3.1 Direct deployment 25
3.2 Deployment through Reverse Proxy 29
4 Trust-based Intrusion Detection System (IDS) 31
5 Moving Target Defences (MTD) 34
5.1 MTD Server 35
5.2 MTD Client 38
6 Trust Message Broker & Trust Components' Integration 40
6.1 Trust Enablers' Integration Overview 40
6.2 IDS integration details 41
6.3 MTD integration details 41
6.3.1 MTD Registration topic 42
6.3.2 MTD Config topic 42
6.3.3 MTD Keep-Alive Request And Keep-Alive Response topics 43
6.3.4 MTD and SAP 44
6.4 Integration & interplay of Security Assurance Platform with the DLTs 44
6.4.1 SAP to DLT record of internally generated Evidence 45
6.4.2 SAP to DLT record of Trust Enabler Evidence 47
6.4.3 SAP to DLT record of Monitoring Evidence 49
7 Additional Security, Privacy, and Trust Primitives 51
7.1 Multi-layer Monitoring 51
7.1.1 Event Captors 51
7.1.2 Events Common Schema Specification 51
7.2 5G Security Considerations 53

7.2.1 Overview of Common 4G/5G Security 54
7.2.2 New 5G-specific innovations 54
7.3 Other best practices, processes, and technologies 55
8 Conclusions 57
References 58
Annex A – IntellIoT Asset Model specification form 59
Annex B – Asset Model Grammar 61

1 INTRODUCTION

This deliverable, being the first output of Task 4.4 ("Trustworthy infrastructure by design"), aims to provide details on the design and development of the first version of the trust mechanisms that are an integral part of the IntellIoT framework. These mechanisms, and the associated enablers, jointly provide trust-by-design, safeguarding the security and privacy of IntellIoT and its target deployment (e.g., use case deployment), ultimately building trust into IntellIoT and the application deployment as a whole. These efforts are aligned with the project's 4th Objective: *"Enable security, privacy, and trust by design with continuous assurance monitoring, assessment and certification as an integral part of the system, providing trustworthy integration of third party IoT devices and services"*.

From a high-level perspective, these mechanisms and associated components can be grouped into three pillars:

- Continuous Security Assurance, centred around the provision of evidence-based assessments of the security and privacy posture of IntellIoT and its deployment;
- Security, Privacy & Trust primitives, including innovative enablers such as trust-based intrusion detection, moving target defences, and Distributed Ledger Technologies (DLT), and;
- Multi-layer Security Monitoring, provided by heterogeneous, purpose-developed event captors that are deployed across IntellIoT deployment layers and which cover core components and processes.

These three pillars and their interplay are visualised in Figure 1.

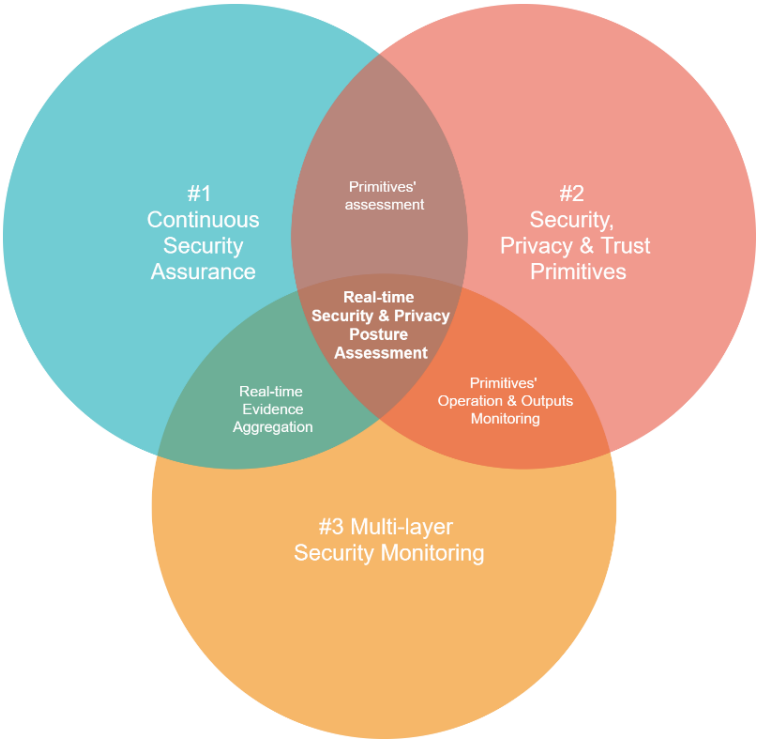


Figure 1. The three implementation-level trust pillars of IntellIoT

From a component development perspective, initial details on the individual enablers comprising the above three pillars, and how they all fit together, have been documented in deliverable D2.3 - "High level architecture (first version)", whereby a separate subsection dedicated to these "Trust Enablers" has been included in all 4 views of IntellIoT's architecture that said deliverable provided. Figure 2 highlights these enablers, as identified in the first version of

IntellIoT's high-level, logical architecture, as documented within D2.3. These WP2 efforts provide valuable inputs for the design and implementation of the corresponding components within Task 4.4.

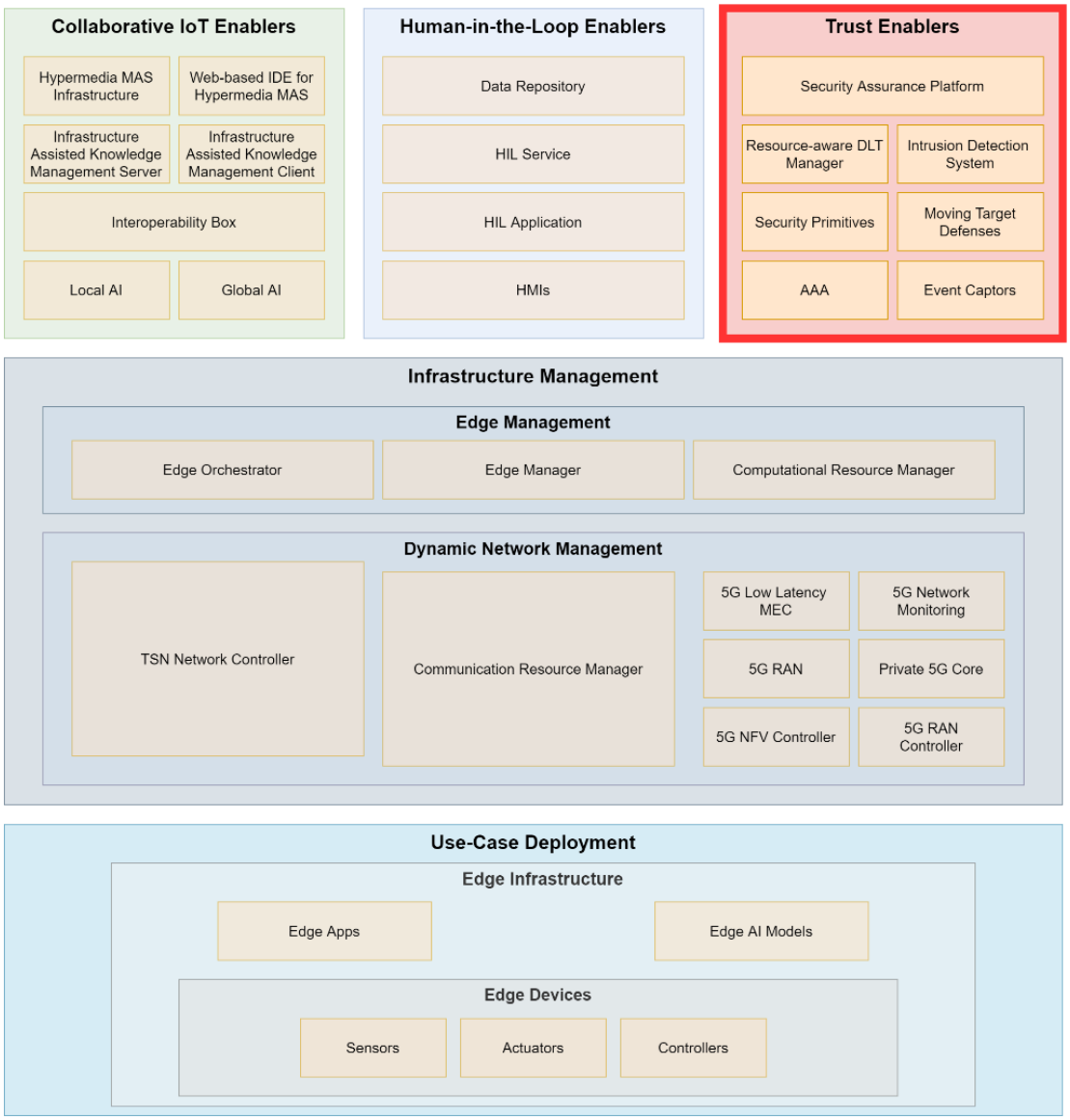


Figure 2. Core trust enablers (top right) within the first version of IntellIoT's architecture

Being the first (i.e., Cycle 1) output of Task 4.4 - this deliverable provides details on the first versions of the trust mechanisms and components, as planned to be integrated into the first release of IntellIoT. Said version is to be delivered through D5.1 – "Integrated IntellIoT framework & use case implementations (first version)", of Task 5.1 ("Integration & implementation"). This is in line with the overall work plan, whereby Task 4.4 outputs its results mainly to Task 5.1.

More details on the interplay of Task 4.4 with other tasks within the project are shown in Figure 3.

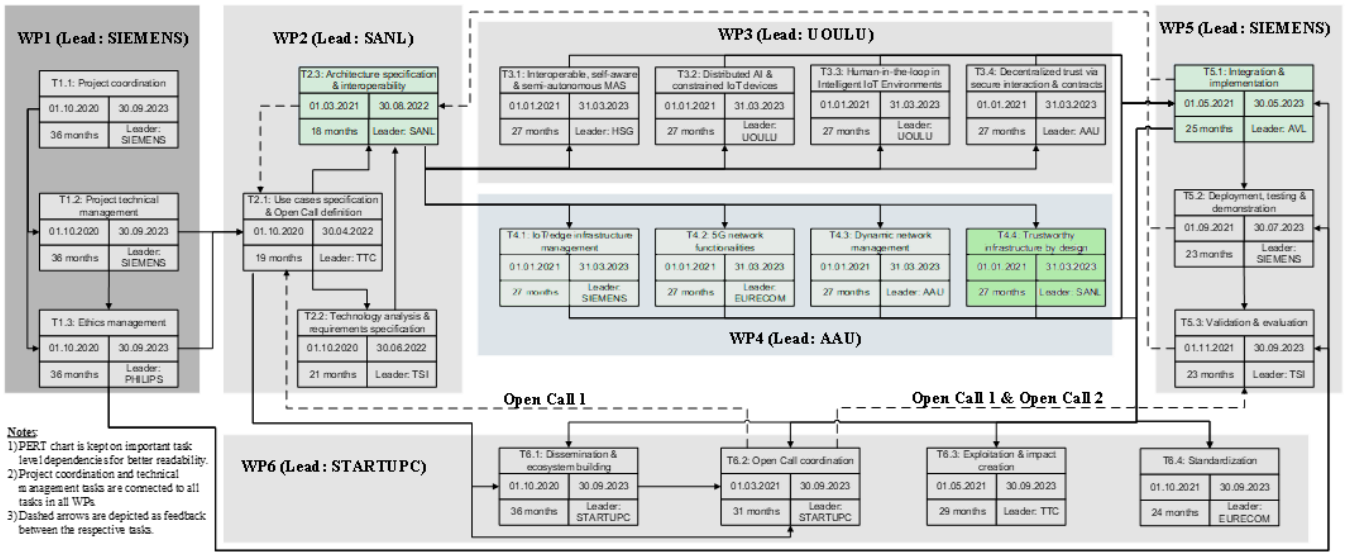


Figure 3. Task 4.4's interplay with other WPs and Tasks of IntellIoT

To present the above building blocks, this deliverable is organised as follows:

- Section 2 presents the Security Assurance Platform (referred to as SAP), i.e., the main enabler of the continuous assurance pillar mentioned above.
- Sections 3, 4 and 5 provide details on core security, privacy, and trust primitives, including the Authentication, Authorisation and Accounting (AAA) solution adopted for IntellIoT, and the novel Trust-based Intrusion Detection System (IDS) and Moving Target Defences (MTD) approaches.
- Section 6 is dedicated to presenting the integration and collaboration between all of IntellIoT's Trust Enablers through the dedicated message broker, including the integration between SAP-MTD-IDS and the integration between the SAP and the DLT -based trust primitives, mostly developed within Task 3.4 ("Decentralized trust via secure interaction & contracts").
- Section 7 documents additional security, privacy, and trust efforts within Task 4.4, including the development of multi-layer monitoring capabilities that enable the corresponding third pillar mentioned above, as well as other security provisions adopted, encompassing 5G security, and the adoption of best practices and state-of-the-art approaches for cryptographic primitives and secure-by-default and fail-secure device configuration.

2 CONTINUOUS ASSURANCE AND CERTIFICATION

2.1 Overview

The Security Assurance Platform (SAP) is an integrated framework of models, processes, and tools to enable the continuous assurance and certification of the security properties of the devices across the IntellIoT infrastructure. It uses different types of evidence to demonstrate the support for the required properties and award the corresponding certificate. These include hybrid assessments that consider different attacks surfaces and attacker capabilities by gathering contextual information (e.g., configuration changes, network, and middleware behaviour), as well as an up-to-date view of all known vulnerabilities across the IntellIoT deployments, from field to backend. The Security Assurance Platform:

- Combines runtime monitoring and dynamic runtime testing to ensure correct and effective operation of security controls
- Can be hooked to different systems programmatically through appropriate probes (e.g., event captors, test tools) to obtain the monitoring and/or test evidence required for assurance and/or certification assessments
- Operates based on models that determine the operational evidence that should be captured from systems and how it should be assessed (e.g., what conditions it should satisfy) to assess the correctness and effectiveness of implemented system security controls
- Enables the runtime assessment of temporal event patterns and rules that can express signature or anomaly-based patterns.

2.2 Internal Architecture

As shown in Figure 4, internally the SAP is comprised of five main modules:

1. **Cyber System Asset Loader**: The component responsible for receiving the Security Assurance Model for the target organization. This model includes the assets of the organization, security properties for these assets, threats that may violate these properties, and the security controls that protect the assets and is based on STS's Assurance Model.
2. **Vulnerability Analyzer**: The Vulnerability Analyzer is responsible to identify known vulnerabilities of assets defined within an organisations' asset model. This component automatically constructs the Common Platform Enumeration (CPE[1], a structured naming scheme for information technology systems, software, and packages) per asset and then retrieves the relevant Common Vulnerabilities and Exposures (CVE[2], a reference-method for defining unique, common identifiers for publicly known information-security vulnerabilities and exposures) entries, by searching in a local copy of the National Vulnerability Database (NVD[3], a U.S. government repository of standards-based vulnerability management data, maintained by the National Institute of Standards and Technology, NIST[4]). This copy is continuously updated by utilising an in-house component that fetches the latest known CVEs from NVD's JSON files.
3. **EVEREST**: A runtime monitoring engine, built-in Java, that offers an API for establishing the monitoring rules to be checked. This module is composed of two submodules: (a) the monitoring database and (b) the monitor.

[1] https://nvd.nist.gov/products/cpe
[2] https://cve.mitre.org/
[3] https://nvd.nist.gov/
[4] https://www.nist.gov/

The role of the module is to forward the runtime events from the application's monitored properties and finally obtain the monitoring results.

4. **Event Captors**: The Event Captor is a tool that, based on collected data and triggering events, formulates a rule or a set of rules and pushes the latter towards the monitoring module for evaluation. Data and events are mostly collected through Elasticsearch[5] based on lightweight shippers (namely Beats), such as Filebeat[6], Metricbeat[7], Packetbeat[8], etc., that forwards and centralizes log data. Data can also be collected through Logstash, an open server-side data processing pipeline that ingests data from a multitude of sources transforms it, and then sends it to Elasticsearch. The Event Captor is initiated through the respective REST calls from the monitoring module.
5. **Dynamic Tester**: The component responsible for initiating the testing assessment. The module consists of two components: (a) the dynamic tester or manager and (b) the dynamic testing tool.

Moreover, the operation of the platform relies on the following databases:

- **EVEREST Database**: Holds the monitoring rules and the overall process done by EVEREST if the templates and other values important for monitor procedure to conclude.
- **Security Assurance Database**: Holds the cyber system asset model, its components, and the results of the assessments.
- **Vulnerabilities Database**: Holds the known vulnerabilities, as retrieved from the NVD database.

Finally, the Assurance tool interfaces with a set of external components, namely:

- **Message Broker (RabbitMQ[9]):** The message broker is a messaging bus that allows communication between the external components and the assurance platform using AMPQ[10] protocol. This will also act as the Trust Broker in IntellIoT; see Sect. 6.
- **OpenVAS[11]:** This is a software framework of several services and tools offering vulnerability scanning and vulnerability management. The tool is used as part of the dynamic testing offered by the Assurance Platform.

[5] https://www.elastic.co/
[6] https://www.elastic.co/beats/filebeat
[7] https://www.elastic.co/beats/metricbeat
[8] https://www.elastic.co/beats/packetbeat
[9] https://www.rabbitmq.com/
[10] https://www.rabbitmq.com/
[11] https://www.openvas.org/

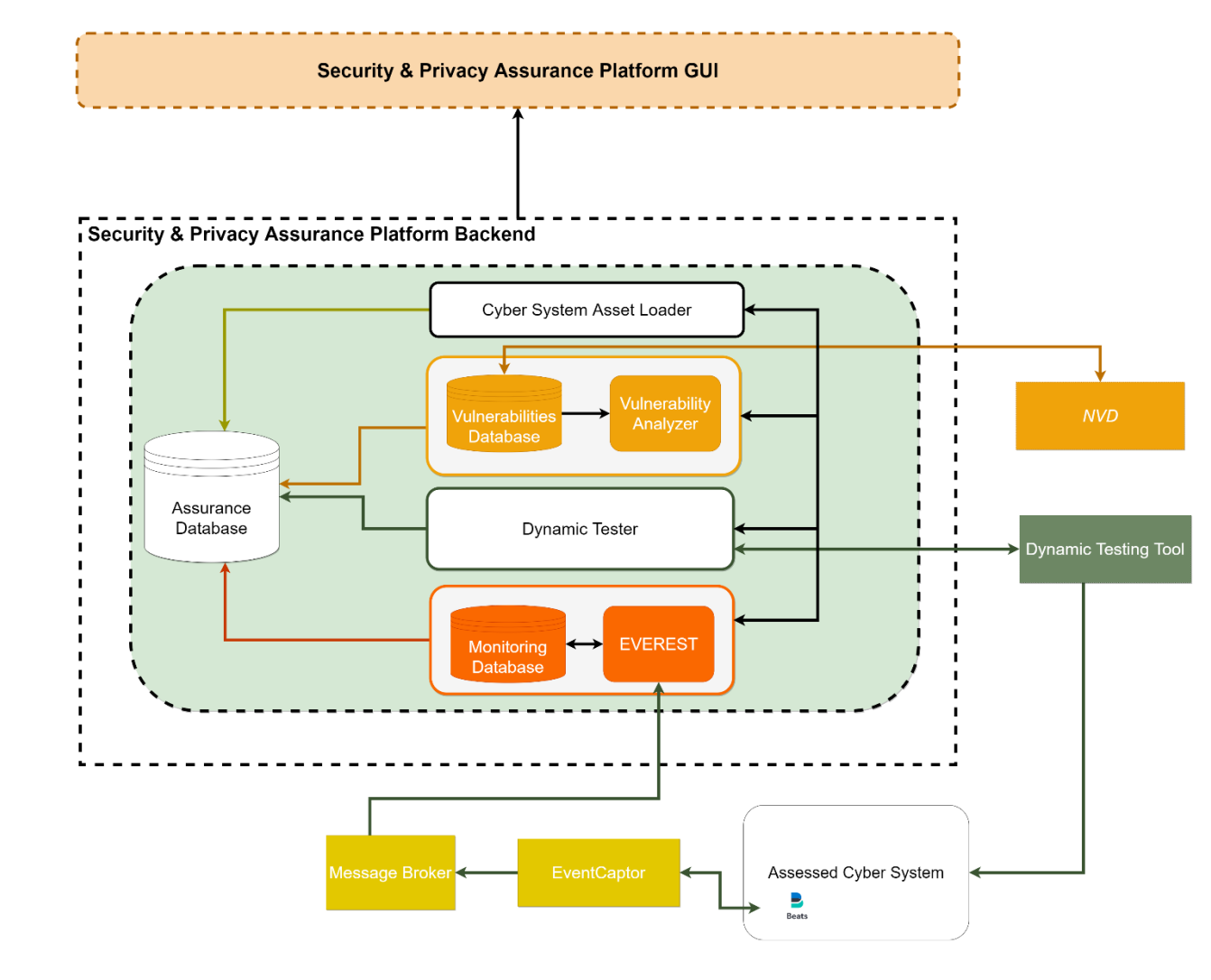


Figure 4. High-level architecture diagram of the Security Assurance Platform

2.2.1 CYBER SYSTEM ASSET LOADER

The Asset Loader module is responsible for receiving the Security Assurance Model, written in the Security Assurance Grammar, for the target organisation. The latter utilises the ANTLR4 parser generator to create the grammar and read, process, execute, or translate the structured text inside it. Lastly, the Model includes the assets of the organisation, security properties for these assets, threats that may violate these properties and the security controls that protect the assets. The internal architecture of the module is shown in Figure 5.

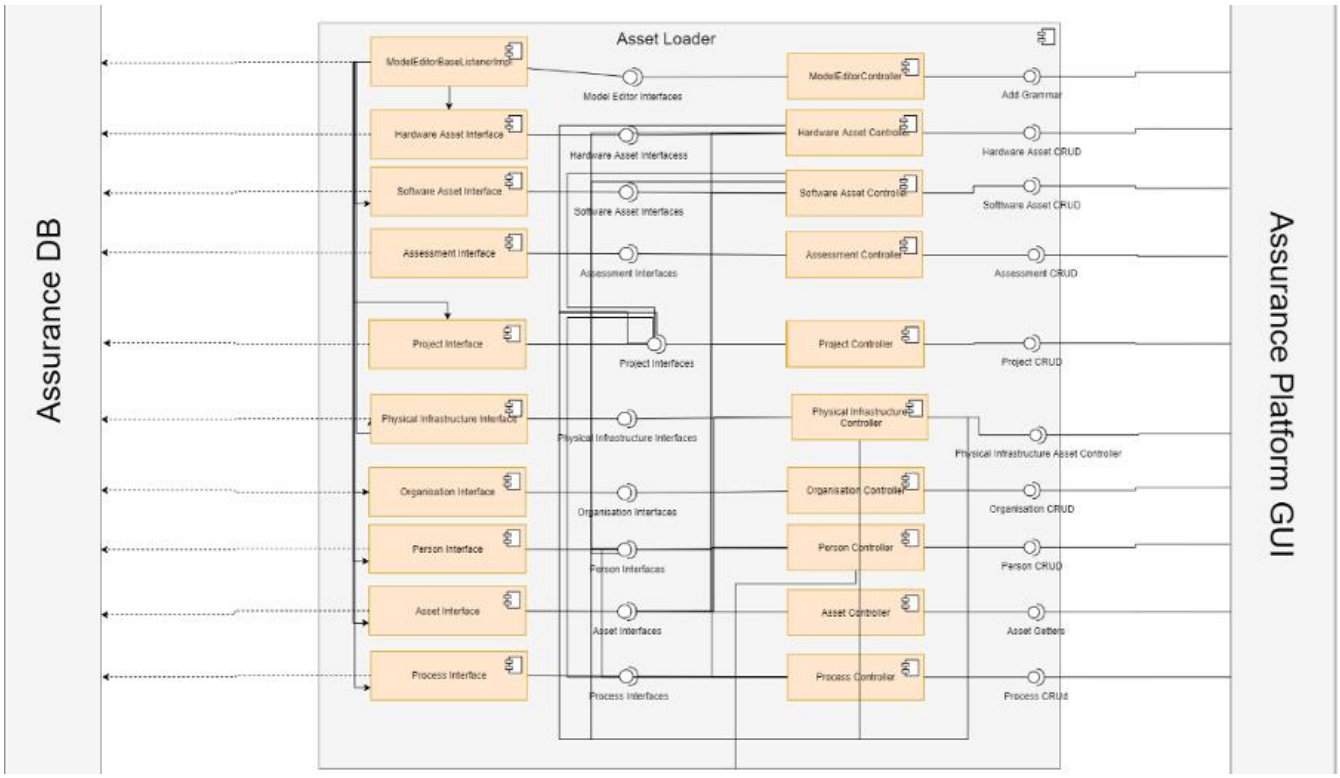


Figure 5. Internal architecture of SAP's asset loader module

2.2.2 VULNERABILITY ANALYZER

The Vulnerability Analyzer module is responsible for loading the known vulnerabilities (of the identified assets) and updating the assurance platform depending on the organisation's assets included in the assurance model. It consists of the following sub-components:

- Job Scheduler
- JSON module
- CPE constructor
- Assessment Controller
- SSE Controller

The JSON module's main functionality refers in getting-updating the CVEs as well as uploading-retrieving them from the Assurance Database. It also retrieves and sets the assessment models and the assessment criterion. The CPE constructor is responsible to construct the CPE for the assets involved in a specific assessment, whereas the Assessment Controller is responsible to initiate a new assessment and check for vulnerabilities in assets based on their CPE. Lastly, the JOB scheduler is responsible of managing the assessment results and updating the CVEs periodically. The internal architecture of the module is shown in Figure 6.

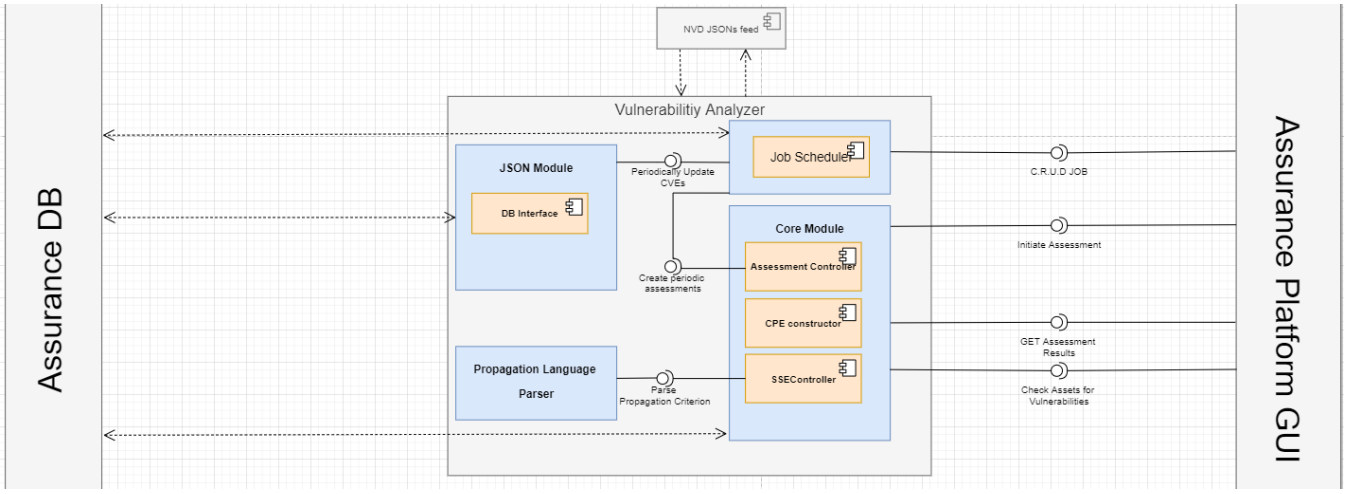


Figure 6. Internal architecture of SAP's vulnerability module

2.2.3 DYNAMIC TESTER

The Dynamic Tester module (see Figure 7) is responsible for initiating the testing and report the results to the assurance database. It consists of the following sub-components: **Model Loader**; **Testing Manager**; **Report Loader**; **Results Uploader** and its REST Interface.

The Model Loader is responsible for translating assessment models to parameters to be utilized by testing tools. Based on this information the Testing Manager is responsible for initiating and scheduling the testing tools to perform the security assessments.

Upon finalization, the results of the security assessments are inputted into Report Loader module which is responsible to interpret them to a format defined by assurance model. The formatted results are then uploaded to assurance database by the Results Uploader module.

Finally, the REST interface module provides the necessary REST functions that are utilized by the security assurance platform GUI to enable the aforementioned functions.

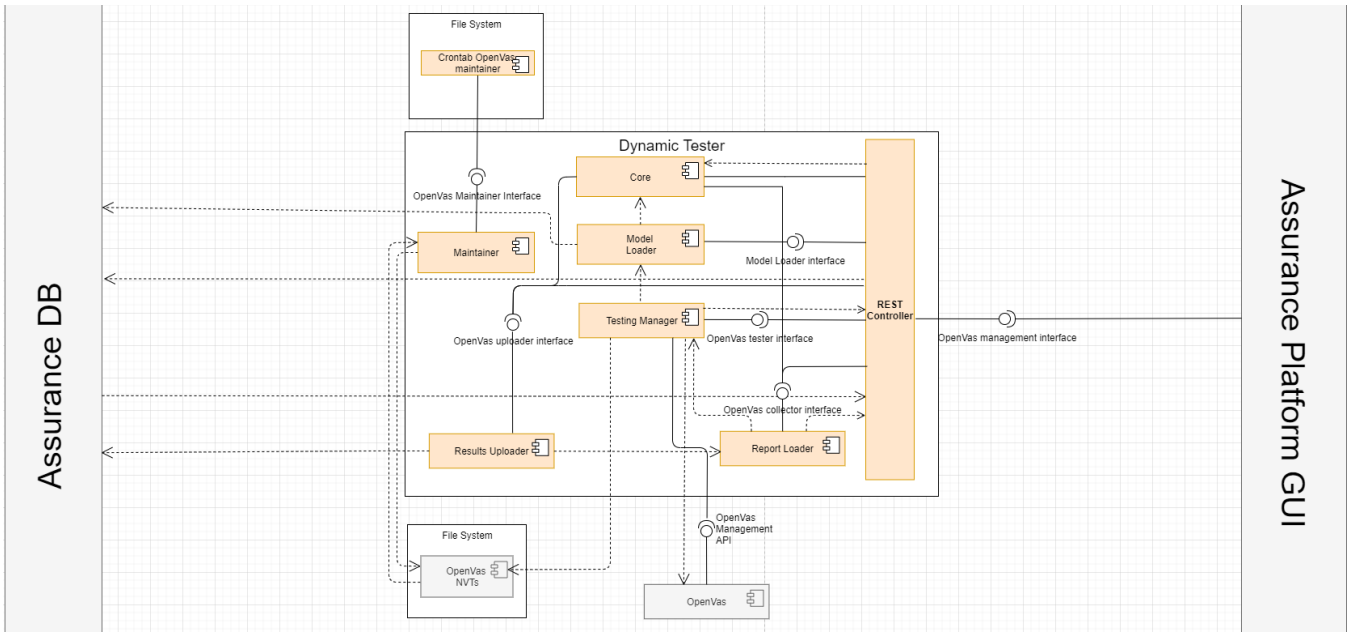


Figure 7. Internal architecture of SAP's testing module

2.2.4 EVEREST MONITORING

EVEREST is responsible for initiating, coordinating, and reporting the results of the monitoring process. This (Java-based) runtime monitoring engine is comprised of two basic submodules (a) the monitoring database and (b) the monitor. It offers an API to other SAP components to instantiate the monitoring rules to be processed. When a pattern is activated through this process, it undertakes the responsibility of checking conditions regarding the runtime operation of the components that implement the pattern. These conditions are specified within patterns by monitoring rules expressed in Event Calculus Assertion (more details on that below). EVEREST can detect violations of monitoring rules against streams of runtime events which are sent to it by different and distributed event sources through the Event Captor Module. It also able to:

- Deduce information about the state of the system being monitored using assumptions about the behaviour of a system and how runtime events may affect its state.
- Detect potential violations of the monitoring rules by estimating belief measures in the potential occurrence of such violations.
- Perform diagnostic analysis to identify whether the events causing a violation are genuine or the result of a system fault or an attack.

The internal architecture of this module is shown in Figure 8. EVEREST interacts with the REST controller that initiates and provides actions for the monitoring assessments. It is implemented in a Docker environment which provides flexibility, portability, and parallel execution capabilities. This component will be able to leverage monitoring mechanisms, such as the Intrusion Detection System (IDS) and Event Captors deployed on IntellIoT's devices (more event captor details are provided in Sect. 7.1) to provide an evidence-based, certifiable view of the security posture of the overall system, with accountability provisions for changes that occur in said posture and the analysis of their cascading effects, supporting the runtime checking based on sets of associated claims and metrics. The methodology and procedures for the automation of the security Monitoring tool is tailored to specific security standard, service level agreements and/or legal and regulatory obligations (e.g., GDPR).

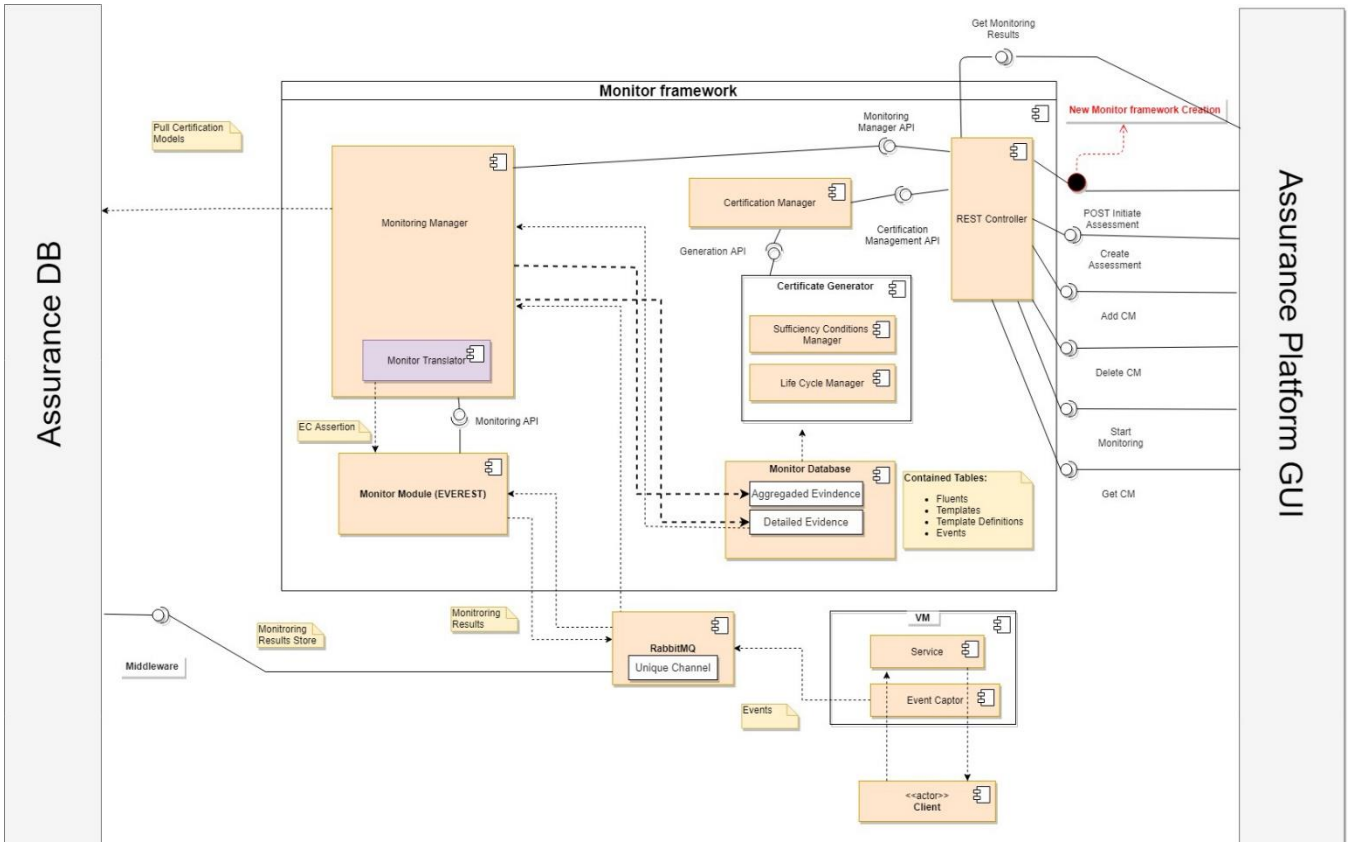


Figure 8. Internal architecture of SAP's EVEREST monitoring framework

The rules and metrics that need to be audited need to be specified within security and dependability patterns (referred to as *S&D Patterns*) using an XML based language, called EC-Assertion. Based on event calculus, EC assertion is a first-order temporal logic language primarily developed not only to represent but also to reason about actions and their effects over time. The basic elements of Event Calculus are *events* and *fluents*. An event in EC is specified as something that occurs at a specific instance of time and is of instantaneous duration. Furthermore, it may cause some change in the state of the reality that is being modelled while this state is represented by fluents.

To represent the occurrence of an event, EC uses the predicate Happens which represents the occurrence of an event `e` that occurs at some point in time `t` within the time range `(t1, t2)` and is of instantaneous duration. The EC predicate `initiates(e, f, t)` signifies that a fluent `f` starts to hold after the event `e` occurs at time `t`. The EC predicate *`terminates(e, f, t)`* signifies that a fluent `f` ceases to hold after the event `e` occurs at time `t`. An EC formula may also use the predicates `initially(f)` and `holdsAt(f,t)` to signify that a fluent `f` holds at the start of the operation of a system and that `f` holds at time `t`, respectively.

EC-Assertion adopts the basic representation principles of EC and its axiomatic foundation and introduces special terms to represent the types of events and conditions that are needed for runtime monitoring. More specifically, given its focus on auditing the operation of software systems at runtime, events in EC-Assertion can be invocations of system operations, responses from such operations, or exchanges of messages between different system components. To represent these types of events, EC-Assertion defines a specific event structure that is syntactically represented by the event term `event(_id, _sender, _receiver, _status, _sig, _source)`.

In this event term:

- *_id* is a unique identifier of the event;

- *_sender* is the identifier of the system component that sends the message/operation call/response;
- *_receiver* is the identifier of the system component that receives the message/operation call/response;
- *_status* is the processing status of an event (i.e., REQ if the event represents an operation invocation and RES if the event represents an operation response);
- *_sig* is the signature of the dispatched message or the operation invocation/response that is represented by the event, comprising the operation name and its arguments/result;
- *_source* is the identifier of the component where the event was captured

To provide an example of an EC monitoring rule, let us consider the case of a Monitoring Rule for Access Control, whereby the access to a webpage's (e.g., patient management web front end) administration page is monitored. Let us assume that said administration webpage is located at `"Home_page/wp-admin/admin_login.php"`. Two rules are created in order to check if the admin services are only executed by admin users, and not by simple users, which could, for example, indicate that somehow a malicious user managed to perform privilege escalation and gain access to said restricted webpage. The two rules and the corresponding triggers would be as follows:

- **Rule 1** – Violation of access control by malicious user
  - **Event Type:** `event_admin_login`
  - **Event_args{}:** `{event_time, John_Doe, visited, Home_page/wp-admin/admin_login.php, 152.185.12.17, simple_user}`
  - **Event Trigger:** `Happens(event_admin_login, event_args{…, User_role}, time) ^ User_role=simple_user` -> **Violation**
- **Rule 2** – Normal access of admin services
  - **Event ID:** `event_admin_login`
  - **Event_args{}:** `{event_time, admin, visited, Home_page/wp-admin/admin_login.php, 172.180.10.15, administrator}`
  - **Event Trigger:** `Happens(event_admin_login, event_args{…, User_role}, time) ^ User_role=administrator` -> **Compliance**

Similar rules can be defined for different monitoring events across the layers of protected infrastructure, from the hardware to the network and to the process (application, operating system, etc.) level.

2.3 The IntellIoT Assurance Model

Being model-driven, at the core of the operation of the Security Assurance Platform is the specification of the associated Assurance Model. Thus, the IntellIoT Assurance Model must be defined, and should holistically cover IntellIoT and its Use Case deployments, in this case.

In more detail, the assurance model should specify all assets of the target cyber system to be protected, known threats that may affect the physical or software components of the system; assumptions regarding the external environment of the cyber system and the behaviour of agents (human- or system-agents) related to it that may affect it (i.e., prevent or introduce threats); and security controls used to mitigate the risks arising from the threats.

The assurance model also specifies assessment procedures, to identify vulnerabilities, assess the proper function of security controls, and to determine how to detect and respond to cyber-attacks. To support this feature, it can also specify any assessment tools that should be employed for the aforementioned assessments prior the deployment of the system (i.e., static analysis and testing tools) or during its operation (i.e., monitoring and dynamic testing tools). Moreover, it specifies parameters determining how the attacks may manifest, how the security controls may respond

to them (e.g., the attack manifestation events captured and detection time, the undertaken response actions) and the outputs that the deployed assessment tools will generate for the situation.

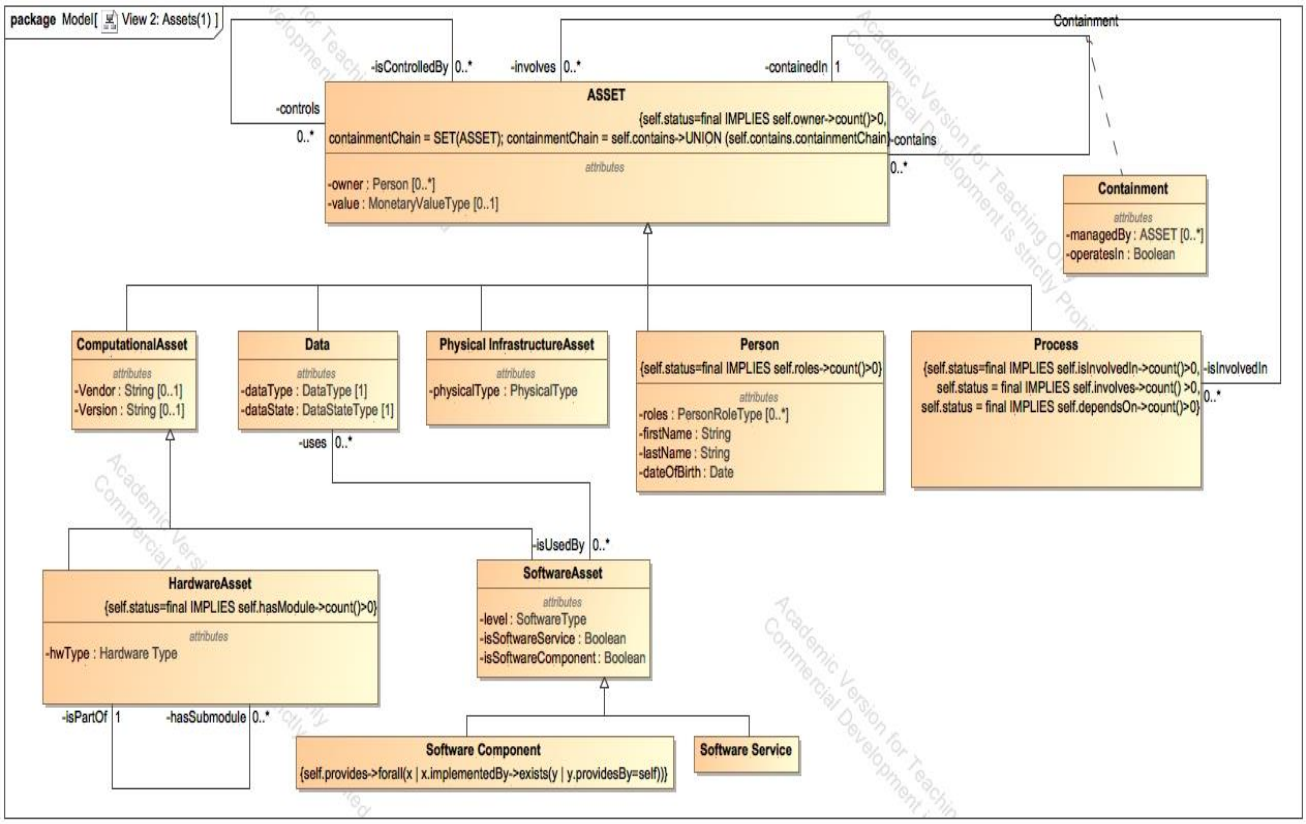


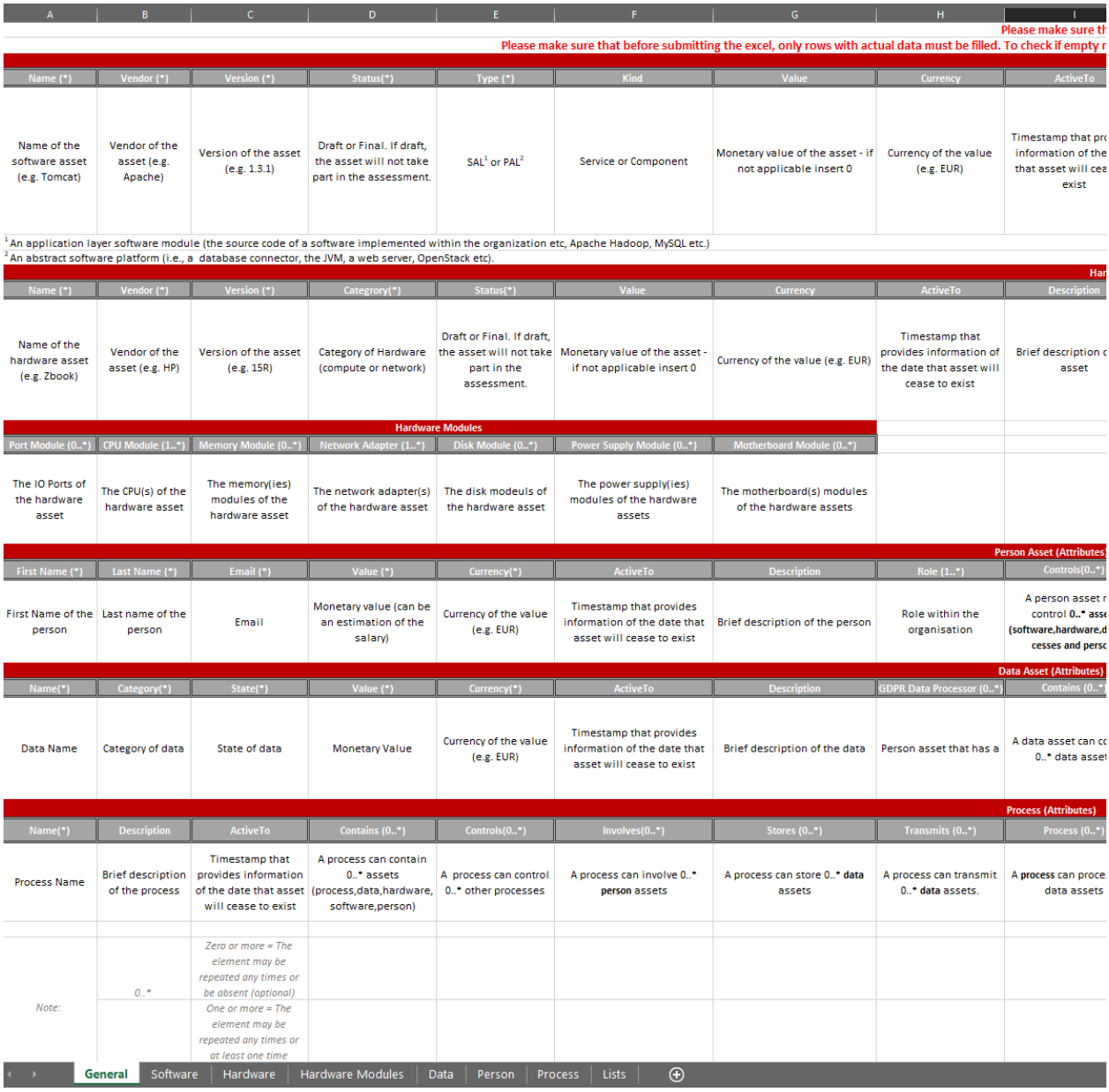
Figure 9. Part of the Assurance Model, showing a view of the Asset sub-model

Since it encompasses all the above, the final model is quite complex - for the sake of brevity, only a high-level description is provided herein. More specifically, the Assurance model consists of the following:

- **Security Assurance Model Element:** The core element of the model, inherited by every other class within the model. Its specification includes the name of the element, the timestamp of the insertion of the element in the assurance platform, the timestamp of the deletion of the element from the assurance platform, a brief description of the element that states its purpose, and the user that inserted this element in the platform.
- **Asset sub-model:** Describes the different assets of the cyber system (i.e., all IntellIoT assets). In addition to the attributes inherited from the Security Assurance Model Element, it also includes the owner (person that owns the asset), its value (in monetary terms) and a number of associations, such as if it is protected by one or more security control(s), if it is contained within another asset and/or if it contains and/or controls other asset(s), and its exposed interfaces. Moreover, the Asset sub-model includes a number of sub-models for different types of assets; namely: *Physical Infrastructure Asset sub-model, Hardware Asset sub-model, Software Asset sub-model, Data Asset sub-model, Person Asset sub-model, Process Asset sub-model and Security Control Asset sub-model.*

A partial view of the Assurance Model, covering the different types of assets in the Asset sub-model and their relationships, is provided in Figure 9.

To create an accurate instance of the IntellIoT assurance model, a specific process is established whereby all partners (i.e., asset owners) are involved. In more details, all Asset sub-model fields needed for assets' specification have been mapped onto tabs and tables of an Excel-based form (see Figure 10 and Annex A – IntellIoT Asset Model specification form). Said file was created to allow all partners to define the needed fields for their assets (IntellIoT components, as involved in each of the three use cases) without having to interact with the Assurance Tool itself.



A	B	C	D	E	F	G	H	I
								Please make sure th
								Please make sure that before submitting the excel, only rows with actual data must be filled. To check if empty r

Name (*)	Vendor (*)	Version (*)	Status(*)	Type (*)	Kind	Value	Currency	ActiveTo
Name of the software asset (e.g. Tomcat)	Vendor of the asset (e.g. Apache)	Version of the asset (e.g. 1.3.1)	Draft or Final. If draft, the asset will not take part in the assessment.	SAL[1] or PAL[2]	Service or Component	Monetary value of the asset - if not applicable insert 0	Currency of the value (e.g. EUR)	Timestamp that pr information of the that asset will cea exist

[1] An application layer software module (the source code of a software implemented within the organization etc, Apache Hadoop, MySQL etc.)
[2] An abstract software platform (i.e., a database connector, the JVM, a web server, OpenStack etc).

Name (*)	Vendor (*)	Version (*)	Categrory(*)	Status(*)	Value	Currency	ActiveTo	Har Description
Name of the hardware asset (e.g. Zbook)	Vendor of the asset (e.g. HP)	Version of the asset (e.g. 15R)	Category of Hardware (compute or network)	Draft or Final. If draft, the asset will not take part in the assessment.	Monetary value of the asset - if not applicable insert 0	Currency of the value (e.g. EUR)	Timestamp that provides information of the date that asset will cease to exist	Brief description asset

Hardware Modules						
Port Module (0..*)	CPU Module (1..*)	Memory Module (0..*)	Network Adapter (1..*)	Disk Module (0..*)	Power Supply Module (0..*)	Motherboard Module (0..*)
The IO Ports of the hardware asset	The CPU(s) of the hardware asset	The memory(ies) modules of the hardware asset	The network adapter(s) of the hardware asset	The disk modeuls of the hardware asset	The power supply(ies) modules of the hardware assets	The motherboard(s) modules of the hardware assets

Person Asset (Attributes)								
First Name (*)	Last Name (*)	Email (*)	Value (*)	Currency(*)	ActiveTo	Description	Role (1..*)	Controls(0..*)
First Name of the person	Last name of the person	Email	Monetary value (can be an estimation of the salary)	Currency of the value (e.g. EUR)	Timestamp that provides information of the date that asset will cease to exist	Brief description of the person	Role within the organisation	A person asset r control **0..* asse (software,hardware,d cesses and persc**

Data Asset (Attributes)								
Name(*)	Category(*)	State(*)	Value (*)	Currency(*)	ActiveTo	Description	GDPR Data Processor (0..*)	Contains (0..*
Data Name	Category of data	State of data	Monetary Value	Currency of the value (e.g. EUR)	Timestamp that provides information of the date that asset will cease to exist	Brief description of the data	Person asset that has a	A data asset can cc 0..* data asse

Process (Attributes)								
Name(*)	Description	ActiveTo	Contains (0..*)	Controls(0..*)	Involves(0..*)	Stores (0..*)	Transmits (0..*)	Process (0..*)
Process Name	Brief description of the process	Timestamp that provides information of the date that asset will cease to exist	A process can contain 0..* assets (process,data,hardware, software,person)	A process can control 0..* other processes	A process can involve 0..* **person** assets	A process can store 0..* **data** assets	A process can transmit 0..* **data** assets.	A **process** can proce data assets
Note:	0..*	*Zero or more = The element may be repeated any times or be absent (optional)*						
		One or more = The element may be repeated any times or at least one time						

General | Software | Hardware | Hardware Modules | Data | Person | Process | Lists

Figure 10. Partial view of the IntellIoT Asset Model specification form (cover page with instructions) – full view in Annex A – IntellIoT Asset Model specification form

In addition to the Excel-based form mentioned above, the asset model can be populated through the SAP front-end (a "wizard" guides the user through the steps), though this is a more time-consuming process, not intended for bulk specification of assets.

A third approach is using the Asset Model Grammar that is aligned with the structure of the underlying asset model. This method enables the bulk specification of assets within the SAP, where using the Excel form is considered inefficient (e.g., large number of assets). An example of an asset definition (in this case an instance of the Ubuntu Operating System) using the Grammar is provided below:

```
SoftwareAsset(vendor("Ubuntu"),version("18.04 LTS"),name("Ubuntu"),kind(Service),type(PAL),project("IntellIoT"),organisation("Sphynx Analytics Ltd."),description("Server OS"))
```

Additional samples of asset definitions through the Grammar are provided in Annex B – Asset Model Grammar.

Once the process is completed, three instances of the IntellIoT Assurance Model will be derived, one for each use case deployment, covering the specific assets involved in said deployment. These will, of course, first reflect the Cycle 1 release and demonstrators of IntellIoT, while a second version of the models will be specified before the Cycle 2 demonstration. The latter will cover both the new/updated components involved in Cycle 2, but also the extended Use Case environments that will be used for the final validation.

2.4 Model-driven Assessments

The Assurance Tool, based on the IntellIoT Assurance Model described in the previous section, performs various types of security assessments based on the corresponding assessment models. More specifically, it supports:

(i) Monitoring Assessments;
(ii) Vulnerability Analysis Assessments;
(iii) Dynamic Testing Assessments;
(iv) Hybrid Assessments, combing (A)-(C) above.

Figure 11 presents the Assurance Tool Assessment page that allows the operator at the SAP (i.e., SAP user) to: (a) initiate a new assessment, (b) upload an existing assessment report and (c) view the existing assessments. Each existing assessment holds a number of general information such as: (a) the assessment model that was utilised (e.g., NVD Vulnerability Assessment), (b) the assessment profile that was used (e.g., focusing on Confidentiality), (c) the status of the assessment (e.g., ongoing), (d) the assessment's initial detection date, (e) how long the assessment is valid (this is mostly used by the monitoring assessment), (f) when the assessment was last checked and (g) the number of current assessment findings.

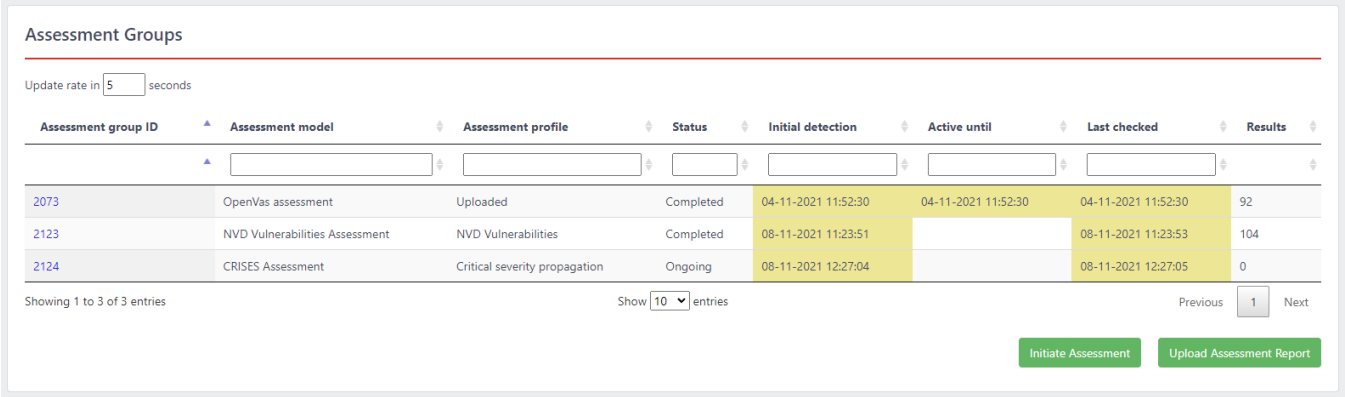


Figure 11. Listing of executed types of Security Assessments on SAP front-end

A user can examine the results of an assessment and some basic information about it by clicking on the "Assessment Group ID" column. As shown in Figure 12, the user is being presented with the basic information of the assessment such as a brief description, the project it belongs to, its creator, the date it was last updated, the tool/service that is

being used, the security properties it checks, its status, the involved assets, and a brief description of the assessment profile. Some key statistics and other information are provided to the user in the form of charts and graphs (e.g., distribution of findings by severity, most vulnerable assets). Further below, the user can see individual assessment results, with some general parameters such as the assessment type, the asset it involves, the security property, the tool likelihood and a universal normalized likelihood, an initial detection timestamp (if applicable), the time it was last checked and the time it ceased to exist (if applicable).

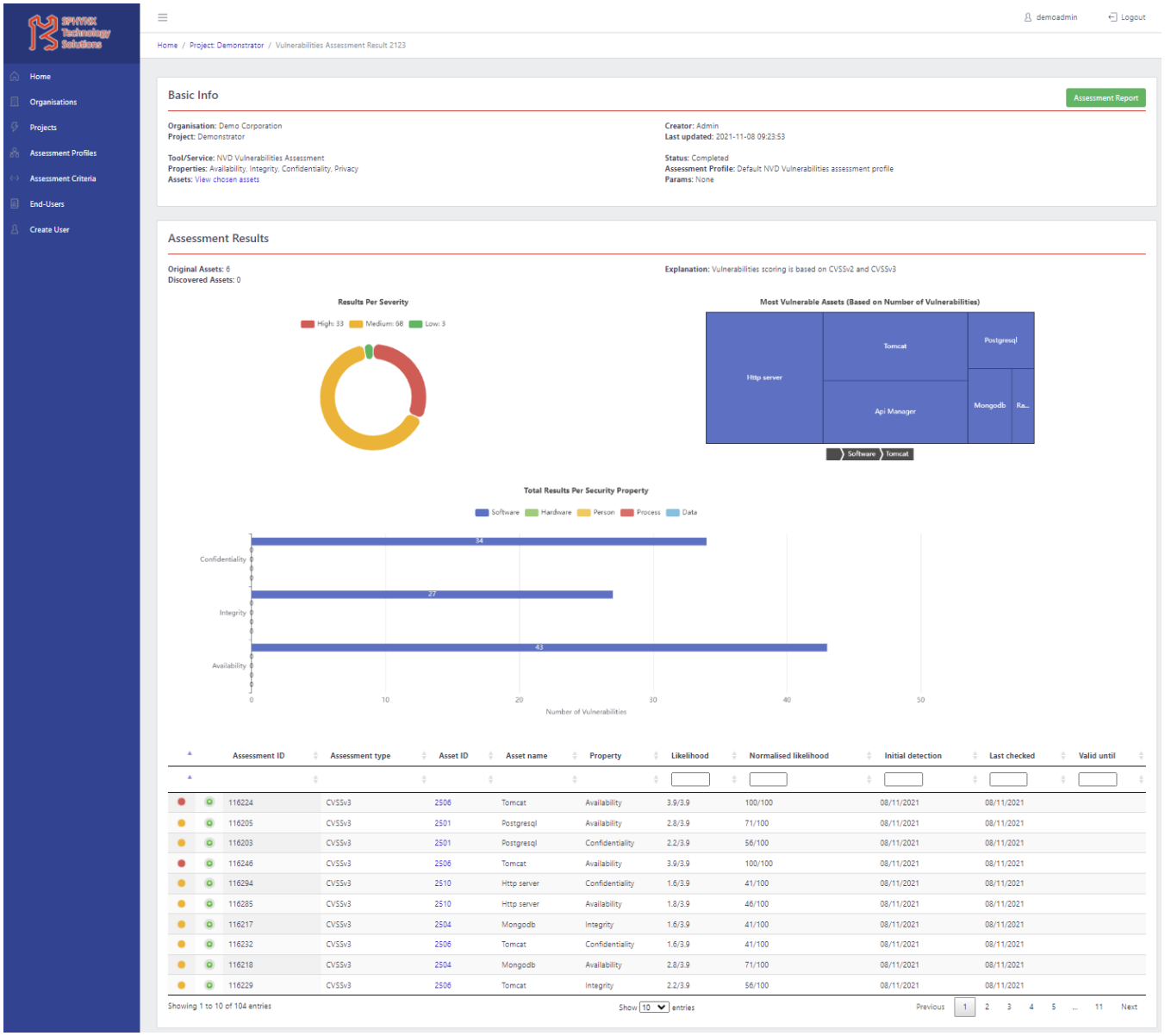


Figure 12. An example of an assessment results screen in the SAP front end.

More specific information is provided in the screens of the different assessment types (i)-(iv), customised to better present the different types of assessment results. For instance, a specific result from a Vulnerability Assessment (i.e., using the Vulnerability Module – see Sect. 2.2.2) is shown in Figure 13, whereby CVE-2020-11993, pertaining to certain versions of the Apache HTTP server is found in the modelled assets. In this case, the presented information follows

the latest version (V3.1) of the Common Vulnerability Scoring System (CVSS[12]), that provides a standardised way to capture the principal characteristics of a vulnerability and produce a numerical score reflecting its severity.

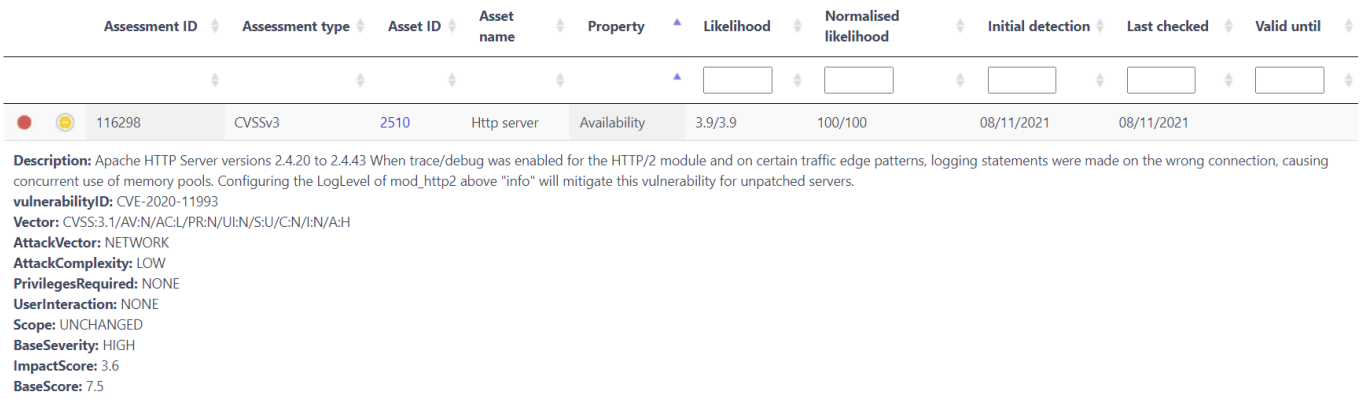


Figure 13. Viewing details of a specific finding of a Vulnerability Assessment

Another example, this time expanding on a specific finding of a Dynamic Testing Assessment carried out using the OpenVAS tool (i.e., using the Testing Module – see Sect. 2.2.3) is shown in Figure 14. In this case, a finding is presented that pertains to the SSH server found during the testing, and the tool identifies a specific vulnerability that stems from a misconfiguration of said server (revealed by directly interacting with the server). Here additional fields are provided, such as how the tool discovered the vulnerability, the confidence in the result (from 0-100, with 100 being that the vulnerability is certainly exploitable), and recommendations to mitigate the finding.

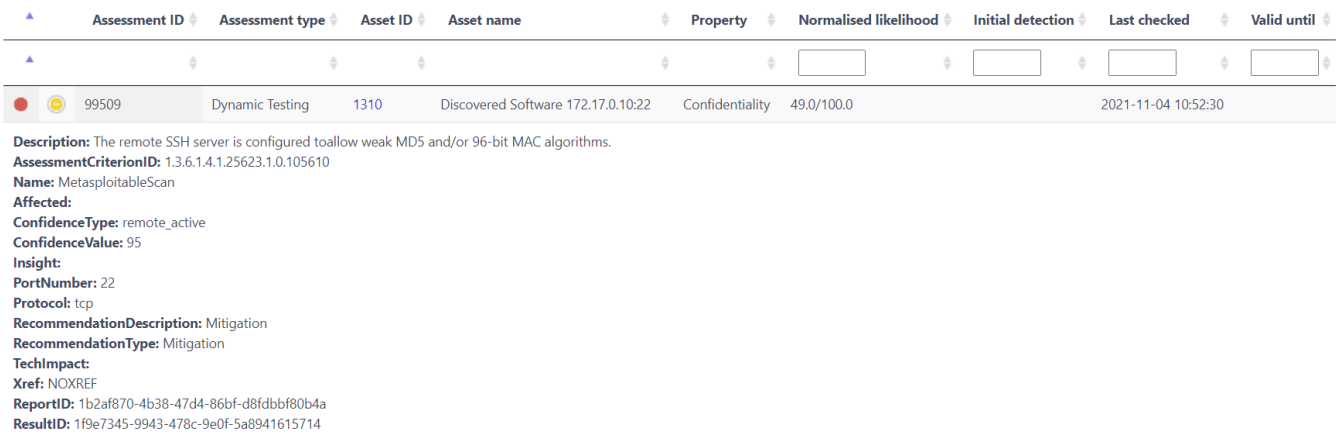


Figure 14. Viewing details of a specific finding of a Dynamic Testing Assessment

Finally, Figure 15 presents details on a finding from a Monitoring Assessment (i.e., using the EVEREST Monitoring Module, as detailed in Sect. 2.2.4). Here the finding is created through the "Violation" of a monitoring rule that was defined to observe for a specific pattern that may indicate ransomware activity. The result provides details on which Event Captor triggered the rule (i.e., where the violation was observed), the exact source and receiver, the time stamp, and other details that allow to uniquely identify the event (and, consequently, the suspicious behaviour), allowing triggering of appropriate mitigation actions, if needed (e.g., through the MTDs), and providing the necessary evidence required for audit purposes.

[12] https://www.first.org/cvss/v3.1/specification-document

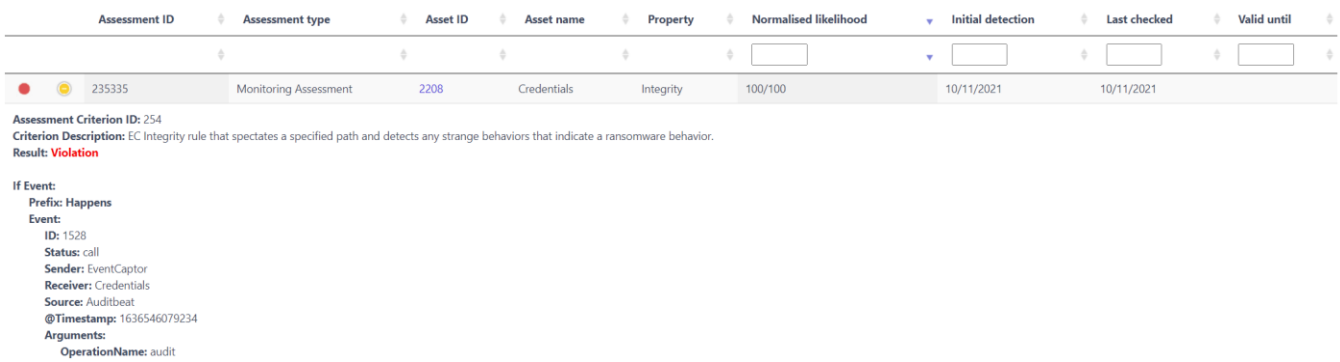


Figure 15. Viewing details of a specific finding of a Monitoring Assessment

It should be noted that Hybrid Assessments are scheduled to be provided in Cycle 2 of the project, so will not be detailed here. A purpose-defined language is being developed to support such Hybrid Assessments that will combine the results from other Assessments (as the ones shown above) based on criteria defined by the user.

3 AUTHENTICATION, AUTHORISATION AND ACCOUNTING

Authentication, Authorization, and Accounting (AAA) in IntellIoT is provided through a framework that allows legitimate users or applications to gain access to protected resources, thus ensuring network security. It can enforce policies (e.g., define multiple permission levels) and audit usage.

The solution selected for IntellIoT is Keycloak[13]. Keycloak is an open-source Identity and Access Management tool for modern applications and services. It uses industry-standard protocols like OpenID Connect[14] for authentication and OAuth 2.0[15] [1] for authorization. This allows for easy integration and well-regarded security. OpenID Connect is an extension to OAuth 2.0.

Keycloak comprises a dedicated server that centrally controls user access. This decouples local security configuration considerations and allows for better scalability and less administrative tasks. For example, if a user is added in Keycloak and configured to have access on some servers within the network, no additional configuration to those servers is needed. The same applies for either adding or revoking user permissions or adding a new resource that some users need to have access to. Furthermore, the servers available on the network are not tasked with storing user credentials locally, which is a preferable approach from both the security and administrative perspectives.

Authorization is the process of evaluating to what extent a user can access a resource. With OAuth 2.0, the following four roles are defined:

- **Resource Owner** (i.e., any entity that owns a number of protected resources, such as files)
- **Resource Server** (e.g., the actual server that keeps the files)
- **Client** (e.g., an application that requests access to a resource from the server is a client)
- **Authorization Server** (the Keycloak server itself in this case)

When a Client wants to access a resource on behalf of an Owner in the Server, it first contacts the Authorization Server. The Authorization Server issues a JSON Web Token (JWT) [3] that gives limited access to the resources, which can then be used to access the Resource Server. JWT tokens are also a standard format based on JSON, supported by libraries for various programming languages[16]. An example id token from authorization is:


```
{
  "exp": 1637007044,
  "iat": 1637006744,
  "auth_time": 1637006744,
  "jti": "fc16ab93-321f-4546-82a7-49f3ecf2b5bf",
  "iss": "http://localhost:8080/auth/realms/myrealm",
  "aud": "myclient",
  "sub": "8007e9f0-db9e-412b-aff8-7775c4088d92",
  "typ": "ID",
  "azp": "myclient",
```


[13] https://www.keycloak.org/
[14] https://openid.net/connect/
[15] https://oauth.net/2/
[16] https://jwt.io/

```
  "nonce": "35aa914f-a761-47d6-814e-be1119b9baec",
  "session_state": "716beddd-14e0-49ca-9081-a6f2a76e881d",
  "at_hash": "9VSilBvnLGVgqtyMYhkHUg",
  "acr": "1",
  "sid": "716beddd-14e0-49ca-9081-a6f2a76e881d",
  "email_verified": true,
  "preferred_username": "myuser"
}
```


Authentication is the process of identifying a user. OpenID Connect defines three roles:

- **End User** (i.e., entity that needs to be authenticated)
- **Relying Party** (the application/server that asks for a user to authenticate himself/herself before accessing it)
- **OpenID Provider** (the server providing the actual authentication, i.e., Keycloak in this case)

Similarly to the authorization case, a JWT token is used. In this scenario it contains additional information in order to signify the authentication process. An example access token from authentication is:


```
{
  "exp": 1637007044,
  "iat": 1637006744,
  "auth_time": 1637006744,
  "jti": "a7b83fea-c54e-4af1-83e9-3ed4e5a2b0c2",
  "iss": "http://localhost:8080/auth/realms/myrealm",
  "aud": "account",
  "sub": "8007e9f0-db9e-412b-aff8-7775c4088d92",
  "typ": "Bearer",
  "azp": "myclient",
  "nonce": "35aa914f-a761-47d6-814e-be1119b9baec",
  "session_state": "716beddd-14e0-49ca-9081-a6f2a76e881d",
  "acr": "1",
  "allowed-origins": [
    "http://localhost:8000"
  ],
  "realm_access": {
    "roles": [
      "default-roles-myrealm",
      "offline_access",
```

```
      "uma_authorization",
      "myrole"
    ]
  },
  "resource_access": {
    "account": {
      "roles": [
        "manage-account",
        "manage-account-links",
        "view-profile"
      ]
    }
  },
  "scope": "openid email profile",
  "sid": "716beddd-14e0-49ca-9081-a6f2a76e881d",
  "email_verified": true,
  "preferred_username": "myuser"
}
```


Accounting is the last functionality provided which is about monitoring authentication and authorization events for auditing. These are clustered in two categories: the login events and the admin events. Login events are generated during normal operation and cover all authentication and authorization aspects. Some basic examples are 'Login', 'Login Error', 'Register' and 'Logout', as shown in Figure 16.

Time	Event Type	Details
11/23/21 1:34:50 PM	LOGOUT	Client: User: 20d1c349-0c15-40b4-9fb2-6fe64b2d5895 IP Address: 172.23.0.1 Details: +
11/23/21 1:34:42 PM	CODE_TO_TOKEN	Client: account-console User: 20d1c349-0c15-40b4-9fb2-6fe64b2d5895 IP Address: 172.23.0.1 Details: +
11/23/21 1:34:42 PM	LOGIN	Client: account-console User: 20d1c349-0c15-40b4-9fb2-6fe64b2d5895 IP Address: 172.23.0.1 Details: +
11/23/21 1:34:41 PM	IMPERSONATE	Client: User: 20d1c349-0c15-40b4-9fb2-6fe64b2d5895 IP Address: 172.23.0.1 Details: +

Figure 16. Login events in the Accounting subsystem of Keycloak

Admin events are generated during Keycloak administration and cover any action that can be performed from within the administration web UI. Example of such events are 'Create/User', 'Create/Group' and 'Action/User' (see Figure 17).

Time	Operation Type	Resource Type	Resource Path	Details
11/23/21 1:34:06 PM	ACTION	USER	users/20d1c349-0c15-40b4-9fb2-6fe64b2d5895/reset-password	Auth
11/23/21 1:32:19 PM	CREATE	GROUP_MEMBERSHIP	users/20d1c349-0c15-40b4-9fb2-6fe64b2d5895/groups/1195c5bb-9621-4b2e-be92-dc1238f8d1c0	Auth Representation
11/23/21 1:32:04 PM	CREATE	CLIENT	clients/b5ad9248-5015-44ea-8a43-90df2faf929b	Auth Representation
11/23/21 1:31:51 PM	CREATE	GROUP	groups/1195c5bb-9621-4b2e-be92-dc1238f8d1c0	Auth Representation
11/23/21 1:31:16 PM	CREATE	USER	users/20d1c349-0c15-40b4-9fb2-6fe64b2d5895	Auth Representation

Figure 17. Admin events (e.g., user creation) as recorded in the Accounting subsystem of Keycloak

Every action can be recorded and stored in the Keycloak database as well as the system log. For example, a system log for a 'Login' event is shown below:

```
Nov 12 11:33:16 mathousalix oauth2-aaa[2010]: 09:33:16,343 DEBUG
[org.keycloak.events] (default task-7) type=LOGIN, realmId=master,
clientId=security-admin-console, userId=8a28a08b-c8f2-4fb2-a34d-2867033b5230,
ipAddress=172.20.0.1, auth_method=openid-connect, auth_type=code,
response_type=code, redirect_uri=http://localhost:8080/auth/admin/master/console/,
```

```
consent=no_consent_required, code_id=ac84a4c0-0e3f-4490-80ec-8d3e0bf9d504,
username=admin, response_mode=fragment, authSessionParentId=ac84a4c0-0e3f-4490-
80ec-8d3e0bf9d504, authSessionTabId=t6PT86TqkU4
```

In the following two subsections, two approaches in the deployment of Keycloak in IntellIoT are presented. The first one is a containerised direct deployment of Keycloak, while the second one uses an intermediate reverse proxy node between a web application and a service that requires authentication and authorization.

3.1 Direct deployment

To facilitate integration with the IntellIoT deployments in the different use cases, Keycloak will be used as a Docker[17] container, allowing to be easily spawned in different use case environments.

An example authentication for an application[18] is showcased in Figure 18. A newly created instance of Keycloak contains only an administrator account and a default Master realm. A realm is a self-contained configuration, isolated from other realms. It aggregates all the relevant information such as the users, groups, roles that comprises a secure environment.

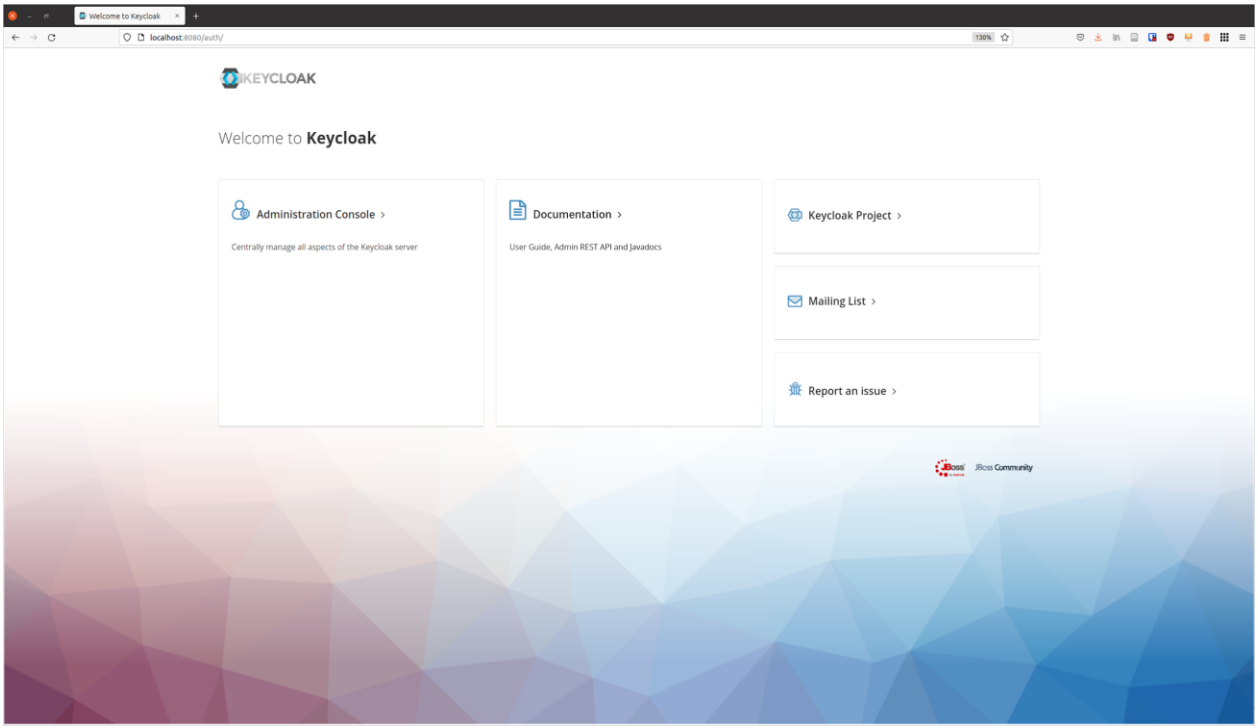


Figure 18. Keycloak homepage

As shown in Figure 19 and Figure 20, a new real, can be created to support the IntellIoT use cases (a realm per use case deployment). We can afterwards start adding all the details that make up the realm. For this example, we have added a group (*iot_group*), a user (*iot_user*), a client (*iot_client*) and a role (*iot_role*). The user is mapped to the group, client, and role. This mapping defines what the user can do.

[17] https://www.docker.com/
[18] https://github.com/PacktPublishing/Keycloak-Identity-and-Access-Management-for-Modern-Applications

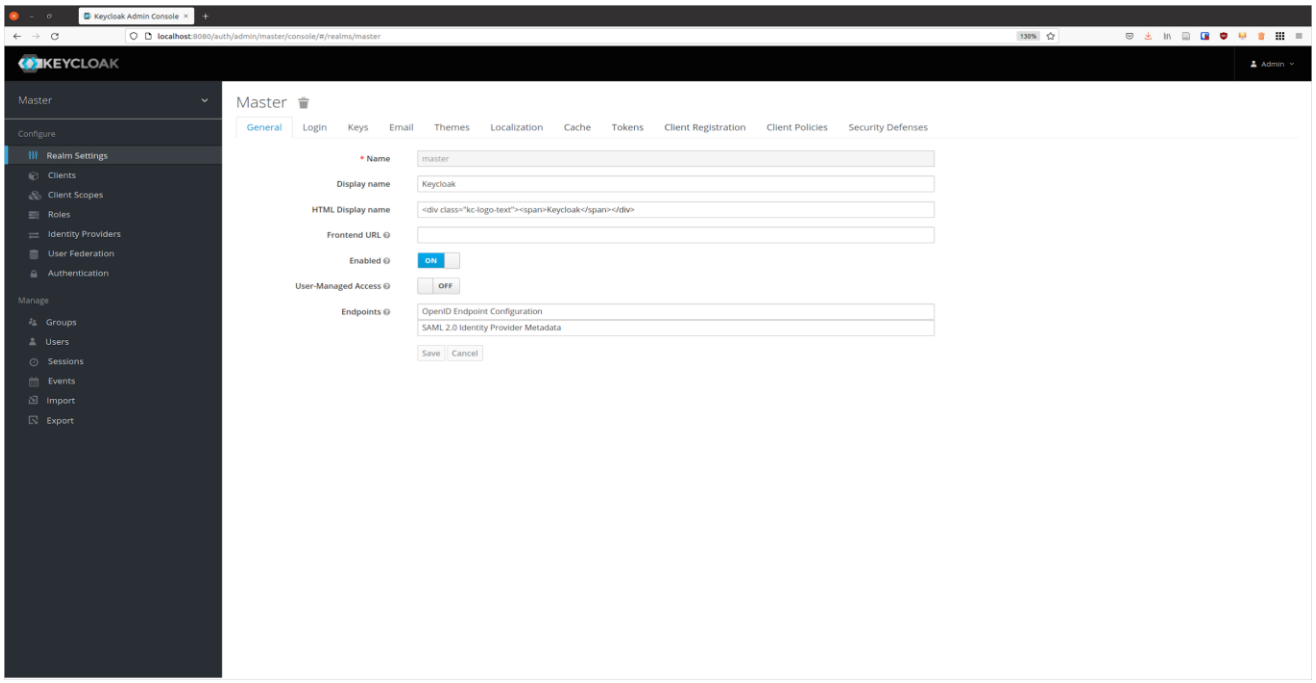


Figure 19. First administrator login to Master realm

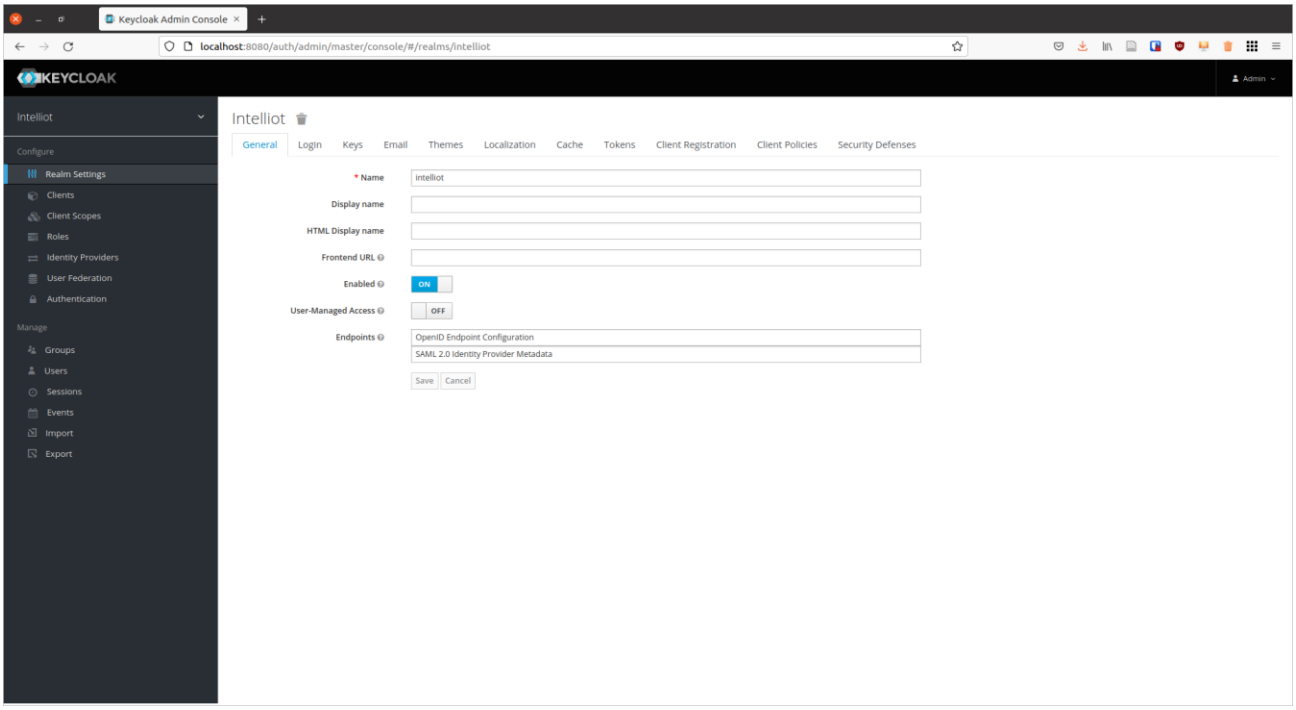


Figure 20. Newly created IntellIoT realm

A sample application is created to test and showcase Keycloak's capabilities (see Figure 21). When the Login button is clicked, a redirect happens to the Keycloak Authentication Server for the user to login (see Figure 22). The application must know where the Keycloak Server is and the realm it needs to get access to.

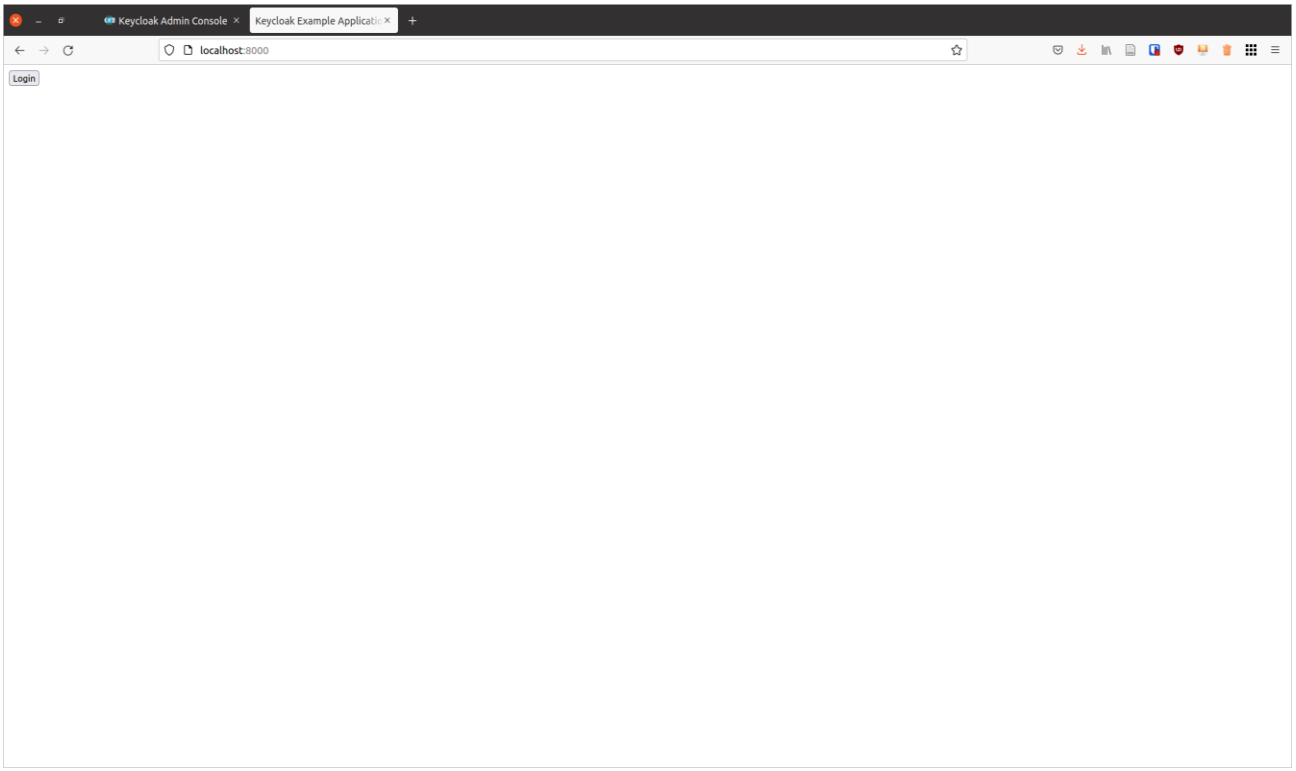


Figure 21. Simple application with login protected by Keycloak

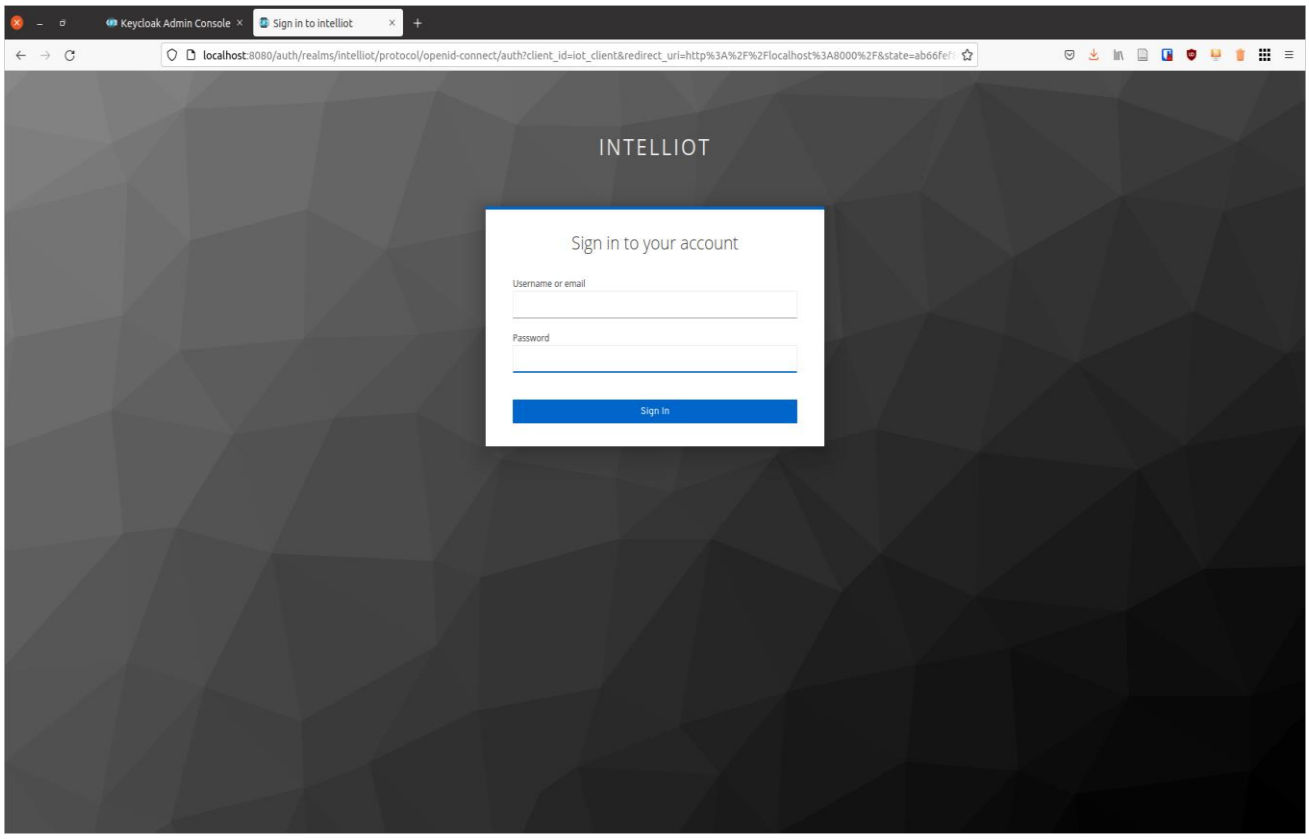


Figure 22. Redirect to Keycloak for authentication

After entering the credentials and Keycloak verifies that the user exists and can use the application, we are redirected back to the application (Figure 23).

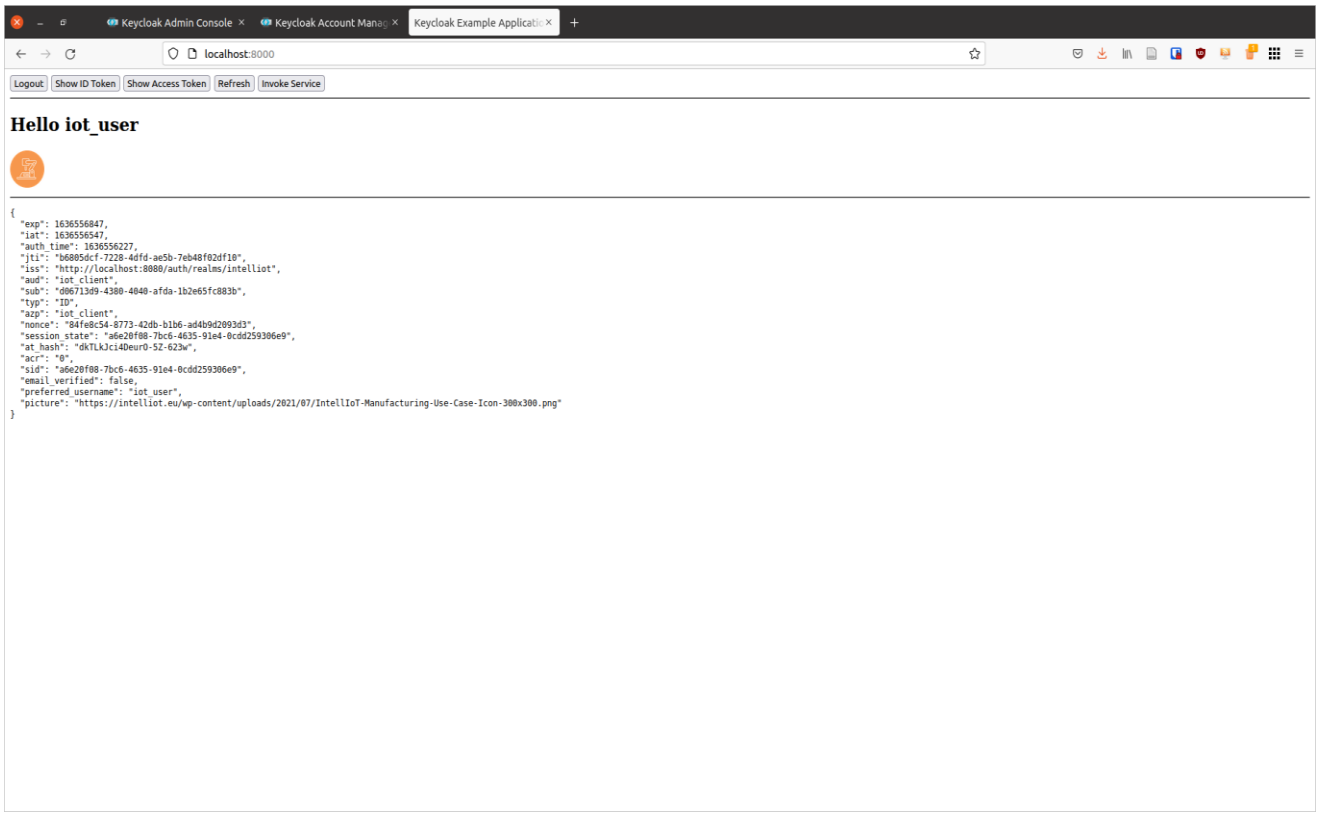


Figure 23. User logged in successfully to test application, via Keycloak

3.2 Deployment through Reverse Proxy

In the example showcased in the previous subsection, the application uses Node.js[19] libraries to properly integrate with Keycloak. This is not always needed or feasible. An alternative scenario is to use a reverse proxy, e.g., Nginx[20], to be integrated with Keycloak.

Having a reverse proxy as an intermediate for web applications that do not have OAuth 2.0/OpenID Connect support can provide authentication and authorization capabilities for them and even simplify the whole architecture of a system.

Figure 24 demonstrates this architecture, where a reverse proxy acts as a middleman between the application and the web service that requires authentication/authorization.

[19] https://nodejs.org/en/
[20] https://nginx.org/en/

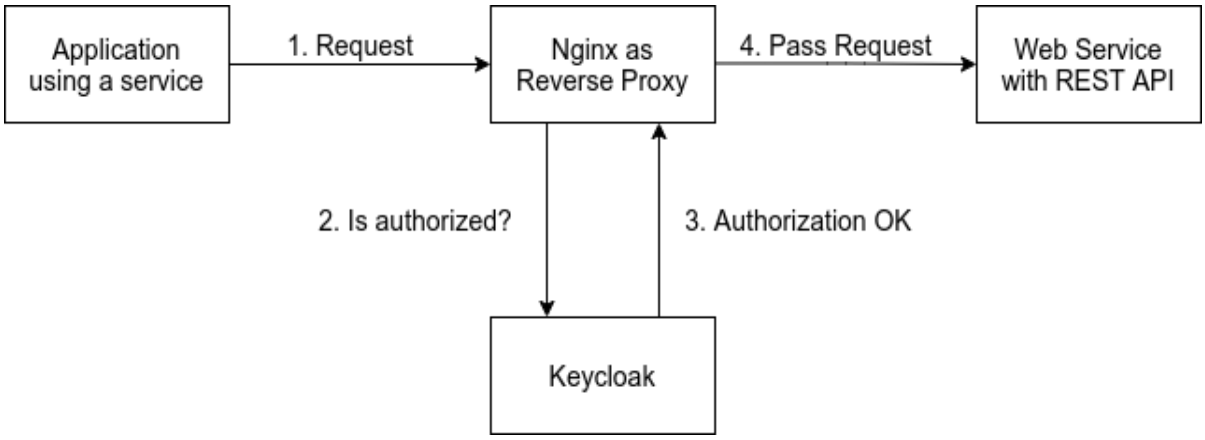


Figure 24. Deployment of Keycloak with a reverse proxy (Nginx).

4 TRUST-BASED INTRUSION DETECTION SYSTEM (IDS)

A Trust-based Intrusion Detection System (IDS) is designed, developed, and integrated into IntellIoT in the form of a software component to be executed on each network node (e.g., drone, tractor, wearable device, robotic arm). More specifically, each instance of the Trust IDS sniffs network traffic and analyses its characteristics in order to assign a trust value for every network node it has communicated with. This system is used to build trust between nodes and assist isolating those that are considered less trustworthy according to certain criteria.

In more detail, when it is first executed, each instance assigns a neutral trust value to all other encountered nodes. After collecting enough data inside a configurable window of time, it computes new trust values based on several filters (trust criteria). Examples of such filters include packet rate and throughput. These filters have configurable thresholds that when exceeded result in a decrease in trust for each associated network node. When thresholds are not exceeded, the trust value of the corresponding node is increased. At the end of every time window, trust values are updated and communicated to the rest of the security components. It is then upon the other security components to take appropriate actions based on the IDS analysis.

The Trust-based IDS deployed in IntellIoT is a custom tool developed in the Go[21] programming language and makes use of only one external library, gopacket[22], which provides packet processing capabilities. The tool runs in Linux OS as a userland application that requires only access to the network interface. A containerised version of the tool is also available.

Figure 25 demonstrates the basic architecture of the tool. At the current (first increment) implementation, the IDS tool supports two configurable filters aiming to detect and prevent rogue nodes that overwhelm the network with unacceptable amounts of network traffic. This also applies to denial-of-service (DoS) attacks that behave in a similar fashion; however, they are instrumented by a malicious actor. The architecture of the tool allows for future integration of additional filters, each of which may have its own parameters independently.

[21] https://golang.org/
[22] https://github.com/google/gopacket

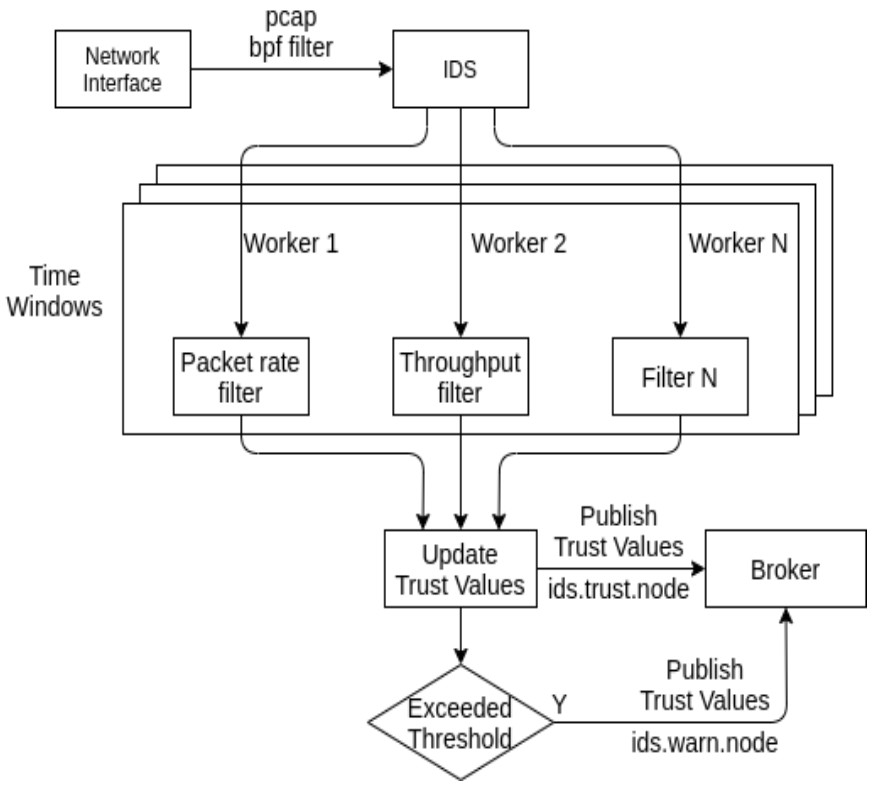


Figure 25. Architecture of the Trust-based IDS tool.

At each time interval, the IDS collects all data recorded for each node and computes a pass/fail value depending on the configuration of the filters (one value for each node). A pass value indicates that the communication with a node is within the acceptable limits set as the filter parameters. Respectively, a fail value indicates that the corresponding node has operated outside of acceptable limits.

The vector of all those values is passed to the Trust Compute component of the tool. For every node on the network, the Trust Compute component starts with a neutral trust value that as mentioned above is raised or decreased according to the behaviour of the node. The computation of trust favours the early and fast detection of malfunctioning / malicious nodes by assigning heavier weights to the fail values received by the filters than the pass ones. Similarly to [7], the fail values have a four to one larger impact than the pass values. This means that the Trust Value of each network node will decrease four times faster than the rate it increases and therefore an erroneous behaviour will be captured rapidly, and the node will need several time windows of proper functioning to reinstate its trust.

The Trust Component will consider a network node as untrustworthy once its computed trust value falls below a certain threshold. In this case, it will trigger a warning message to external security modules (such as the MTD server and SAP) indicating that priority action should be taken. Otherwise, the computed trust values are periodically transmitted to the external security modules.

Two important remarks need to be made. First, the process of declaring a node as untrustworthy masks random events that may be attributed to a momentarily out-of-expected behaviour - a single event cannot render a node as untrustworthy. This is accomplished by using a configurable threshold below the neutral trust value in order to declare a node as untrustworthy. Second, at the first iteration of the tool, this threshold value is part of the user configuration of the tool, and it is set using a manual process or some prespecified heuristic. At the second iteration of the tool, we

plan to introduce more capabilities or dynamic analysis functionalities in order to determine the value of the aforementioned threshold.

The Trust-based IDS instances are executed on each network node we need to analyse its network traffic. As mentioned above, there exists a common secure channel, which is an Advanced Message Queueing Protocol (AMQP) broker, that is used to send the trust values and warning messages to the other security components (the MTD module, detailed in Sect. 5, and the SAP, detailed in Sect. 2). AMQP is an application layer messaging protocol that allows for various routing topologies, such as point-to-point and publish-subscribe, with TLS support to provide secure communication. This communication scheme is depicted in Figure 26. More details on the AMQP Trust Broker are provided in Sect. 6.

The integration between the Trust IDS and the rest of the Trust Enablers is shown in Figure 26. More details on the integration approach are also provided in Sect. 6.

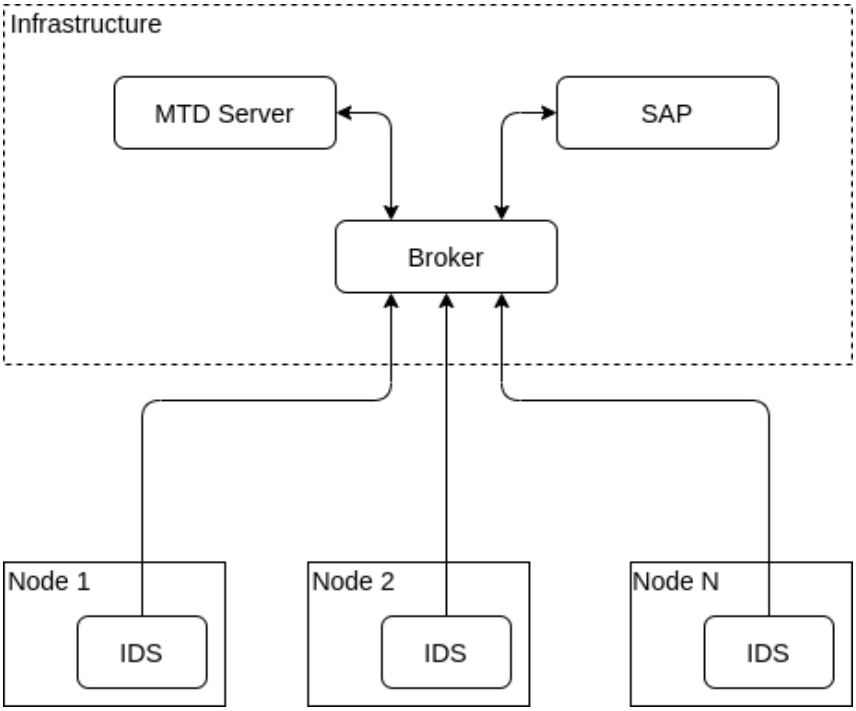


Figure 26. Communication between IDS components and the MTD and SAP modules

5 MOVING TARGET DEFENCES (MTD)

Currently, network infrastructures have adopted mostly static configurations, and rely on traffic monitoring and perimeter defence techniques (for example, Firewalls and Intrusion Detection Systems) to identify and mitigate attacks. But given enough time and resources, the aforementioned defences can be breached.

A new approach is Moving Target Defence (MTD), which is based on the notion that if the defending infrastructure has a dynamic and constantly shifting configuration, attacking it and exploiting vulnerabilities becomes much harder. The attacker now has to 'chase' a constantly moving target, uncover possible vulnerabilities and exploit them before they are rendered invalid by an upcoming configuration change. To make the situation even worse for the attacker, the configuration change is not random, but guided by traditional defences like an Intrusion Detection System, which means that over time, the system becomes more secure against an incoming attack. While in traditional systems with static configurations the difficulty of an attack goes down with time, with MTD it goes up.

MTD can be applied in several different ways, such as changing operating systems, address space, network configuration etc. In IntellIoT, we focus on managing the network configuration and dynamically changing it to fit the needs of the infrastructure. Examples of our strategies include the periodic shuffle of IP addresses and communication ports, as well as the encryption of network traffic while changing in different time intervals (or as a response to certain events) of the encryption keys and the encryption algorithms used to counter sniffing and Man-In-The-Middle attacks. The basic principles of our MTD architecture have been described in deliverable D2.3. In the following subsections we re-elaborate those main concepts for reasons of completeness and then proceed with details about the actual implementation of the MTD infrastructure.

In IntellIoT, the MTD we are developing is a client-server set of software components that manage the network configuration of the edge network. The MTD server is responsible for managing all the clients, handling events like warnings from the IDS or actions from the HIL and generating new configurations. The MTD clients are responsible for applying the configuration sent by the server, and based on that configuration, encrypt the traffic, and transmit it through tunnels (PPTP [4] with IPSec [5]) to avoid packet sniffing and Man-In-The-Middle attacks.

The server and clients communicate by exchanging the network configuration. Parts of the network configuration are the same for all clients, and parts are specific to each client.

The configuration has the following format:

```
type VPNConfig struct {
    //TUN
    CIDR string
    //UDP
    Port   int
    Protoc string
    //Cipher
    CipherKey  string
    CipherType cipher.CipherType
    //router
    ExtIP      string
    IntIP      string
    RoutesToAddmap[string]string
```

```
    RoutesToRemove []string
}
```

The CIDR, ExtIP and LocIP are specific to each client, while everything else is common for all clients.

The way our MTD works is that in each client we provide a new network interface (*mtd0*) with its own IP address, named internal IP (*IntIP*). That means that each client has a pair of IP addresses, the IP address with which it communicates to the network, named external IP (*ExtIP*) and the internal IP. Such a pair is considered a route. When an application uses the mtd0 interface, the packets that are generated have the internal IPs as source and destination, and when trying to send the packet to another client, we first search the destination to find the corresponding external IP. If the route is found, we then encrypt the packet and encapsulate it in a UDP packet with the external IPs as source and destination.

5.1 MTD Server

The MTD server proactively generates a new configuration at fixed intervals to shift the attack surface, and when a warning is received, it reactively generates a mitigation configuration based on predefined strategies to mitigate the possible attack. It is also possible for the HIL service to send actions (e.g., *isolate node X*) through the Security Assurance Platform. Configuration changes focus on network layer 3 and above, and do not interfere with the TSN controller and the resource reservations it maintains.

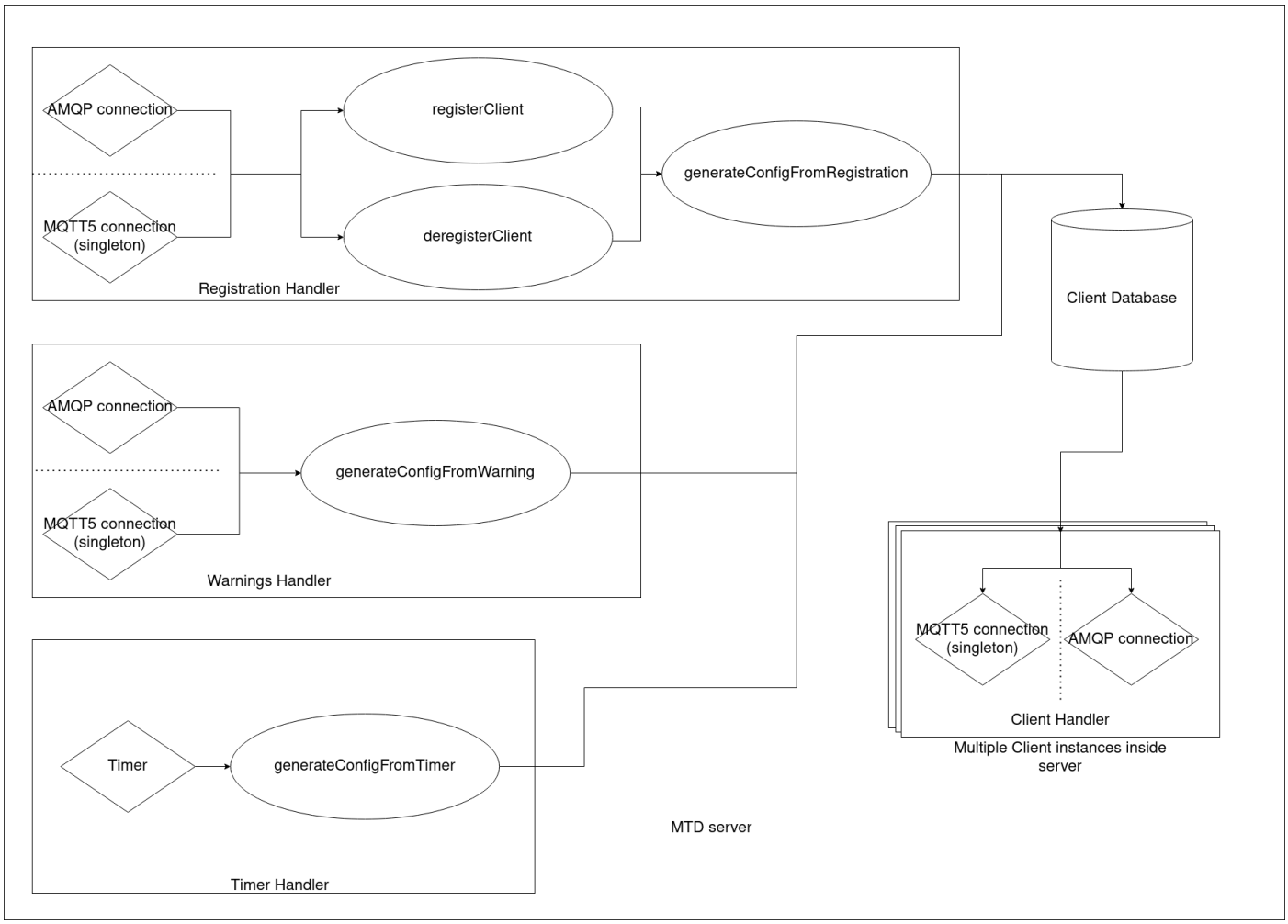


Figure 27. MTD server Architecture

The MTD server contains three main handlers, a client database, and multiple instances of a fourth handler. Based on a static configuration provided at start up, the MTD server connects to the message broker (see Sect. 6).

The first main handler is the registration/deregistration handler. This handler manages registration requests from new clients and deregistration warnings from existing clients. More specifically:

- When a new client requests to register, it provides its current IP address and a client name. The client's name is automatically generated by the MTD client, it is later used to connect to the message broker and must follow a specific format that we will see in the client section. Both the IP address and the client name are checked and if they conflict with an existing client the registration is rejected. If there are no conflicts, a client handler is created, and the client is added to the internal client database.
- When an existing client deregisters, the client handler is removed from the database and stopped. After any changes to the database, a new configuration is generated and sent to all clients. The new configuration has an IP pair assigned to the new client in the case of a registration and includes a new routing table that either includes the newly registered client or excludes the deregistered one. All other parts of the configuration remain the same.

The second main handler is the warnings handler. Every message that comes from an external source (e.g., HIL, SAP, IDS etc) is handled as a warning. Warnings have the following format:

```
type warning struct {
    Type   string
    Action string
    Body   string
    Time   *timestamppb.Timestamp
}
```

The available types are "info", "warning" and "error". The available actions are "drop" to drop an active client and "allow" to restore connectivity to a dropped client. Also, the Body variable contains information relevant to the type and action of the warning. Finally, a timestamp is included for logging purposes.

Currently, the only supported type-action pairs are error-block to block a client and error-allow to restore one. When we receive an error-block message, we read the body for identification about the client to be blocked and search the database. If we find the client, we set it as dropped and regenerate a configuration with that client removed from the routing table. Similarly, when we receive an error-allow message, we search for the client, unset the dropped flag and regenerate a configuration with that client added to the routing table.

Future work is to add strategies for known attacks and based on the warnings we receive, generate configurations that do not necessarily drop a client, but mitigate attacks in other ways as well.

The third main handler is the timer handler. This handler generates a new configuration at fixed intervals to shift the attack surface. Currently, when the internal timer sends a tick, one of three possible changes is performed:

- Update the UDP (external) connection configuration
- Update the TUN (internal) connection configuration
- Update the encryption

When updating the UDP connection, we change the external IP address of each node, and the port that all nodes use to exchange encrypted traffic. The outcome of this change is that all clients are shifted, and an external attacker that might have found a vulnerability in a specific client would have to start searching for that client again, and a Man-In-The-Middle attack would have to scan the network again to find the new port.

When updating the TUN connection, we change the internal IP address of each node. The rationale is that by changing the routing table we make it harder for attackers to gather information about the topology of the system, even if they gain access to one of the clients.

Finally, when updating the encryption, we choose from a list of predefined algorithms:

- No encryption
- No encryption with snappy compression
- AES CBC encryption
- AES CBC encryption with snappy compression
- AES CBC encryption with HMAC for validation
- AES CBC encryption with HMAC for validation and snappy compression

The AES CBC cipher supports 128/192/256 bit encryption, so we choose a size and generate a random key [8][9].

Future work is to expand the configuration generation algorithm to change more than one option per interval, taking into account the previous state of the system, as well as security levels etc (higher security level would mean more secure-computationally expensive encryption), as well as to shift only a subset of the clients and so on.

The final type of handler is the client handler, one of which is spawned for each registered client. The client handler has the state of each client as assigned by the server, as well as a keep-alive connection. At fixed intervals, each client handler sends a keep-alive request to the client, and the client has to respond within a specific time interval, or it is considered disconnected and removed from the client database.

Finally, in the future, a connection to the TSN controller will be added, to send isolation requests.

5.2 MTD Client

The MTD clients are responsible for managing the network configuration, maintaining an encrypted connection between each other using the routing table sent by the MTD server and applying any changes the server sends.

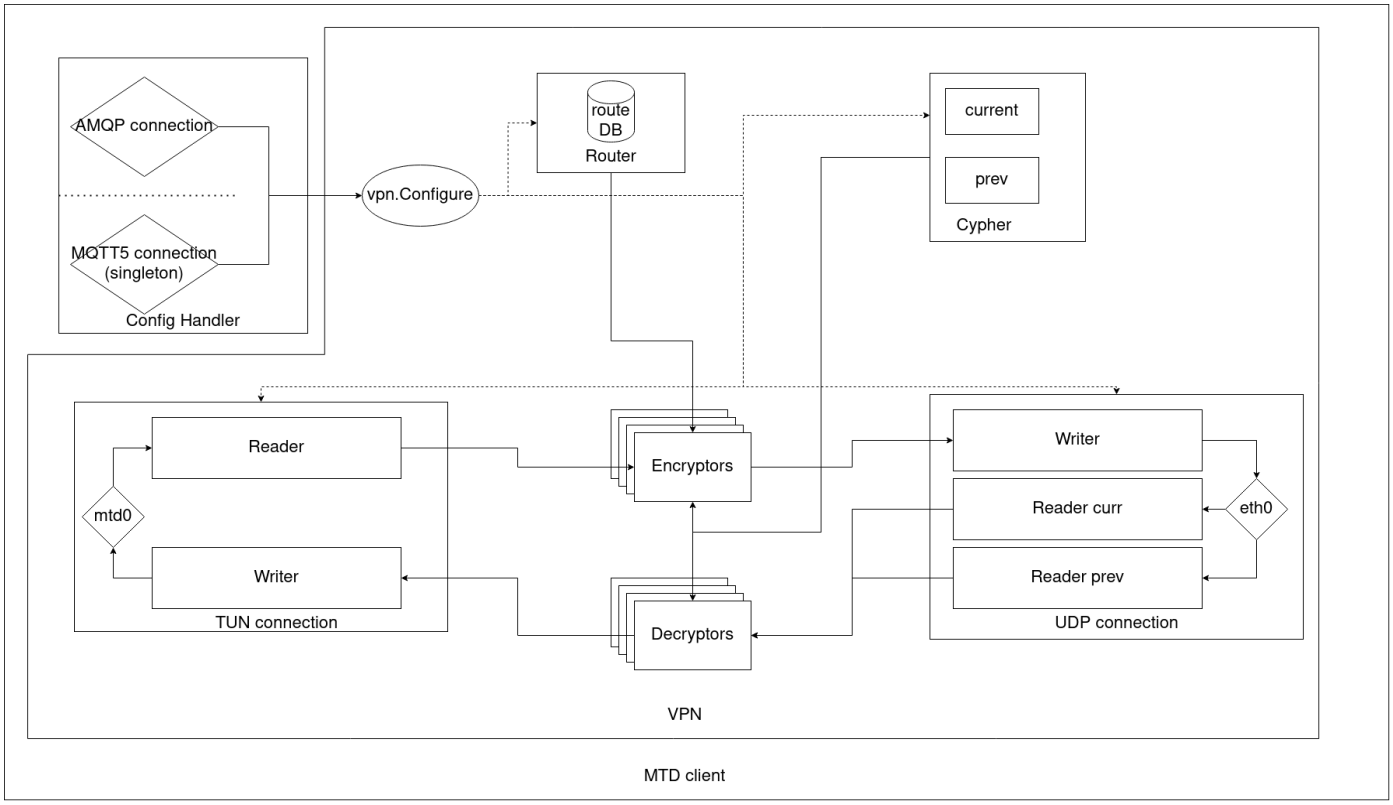


Figure 28. MTD client Architecture

The MTD client contains a config handler, and a VPN module. The VPN module is comprised of:

- A TUN connection and a virtual ethernet interface
- A Router with an internal routing table
- A cipher
- Several Encryptor and Decryptor workers
- A UDP connection using the active network connection of the device

The config handler is responsible for receiving the configuration from the server and sending it to the VPN module. The VPN module then breaks the config into parts and sends each part to the corresponding submodule.

The TUN connection receives unencrypted packets from the application layer to encrypt and send to other clients, and packets from other clients after they have been decrypted to send to the application layer. At initialization a virtual network interface is created (mtd0) and after registering with the server, an internal IP address is assigned to it. A writer and a reader are opened on the interface that are responsible for transferring packets to and from it.

The Router is responsible for keeping the routing table up to date based on the configuration sent by the server and searching the routing table to resolve routes when requested by the encryptors.

The cipher is responsible for keeping the active algorithm and key in sync with the server. At any point there are two states of the cipher, the current and the previous state. The current state is used when encrypting packets and the default state when decrypting packets. If the decryption with the current state fails, we fall back to the previous state. That is expected behaviour when an update to the cipher occurs, since in-flight packets encrypted with the previous key and/or algorithm might reach the client after the update. The previous state is kept active for a limited period of time after each update.

The encryptor workers receive unencrypted packets from the TUN connection, request a route from the routing table and if a route is found, use the cipher to encrypt the packet and send it along with the route to the UDP connection. Multiple encryptor workers can be spawned based on the static configuration of the client, the needs and the available resources of the device, to leverage parallelism in order to reduce the latency incurred by the encryption.

The decryptor workers receive encrypted packets from the UDP connection and use the cipher to decrypt them and send them to the TUN connection. Similarly to the encryptor workers, multiple decryptor workers can be spawned.

The UDP connection is the opposite of the TUN connection. It receives encrypted packets from other clients to decrypt and send to the application layer and sends packets from the application layer to other clients after they have been encrypted. After registering with the server, every time an encrypted packet is sent from an encryptor along with the route, that packet is encapsulated in a UDP packet and sent to the destination using the port specified by the server. Similarly, every time a packet is received from the port specified by the server, that packet is sent to the decryptors. The UDP connection keeps two readers active, one for the current configuration and one for the previous. The reason is the same as for the cipher, to be able to receive packets that were in flight when the update occurred.

At start up, each client has to register with the server by sending a registration request containing the client name and IP address through the broker. The client name is a combination of the user name of the user that owns this device and the devices client ID in the format *<username>_<clientID>* (e.g. ,TSI_sensor1, TSI_sensor2, SANL_tractor etc).

Finally, each client opens a keep-alive connection to the server and awaits for keep-alive requests from the server. When a request is received, the client responds with a status OK message. If the server does not send a request after a fixed amount of time, the connection to the server is considered to be lost.

More details on the integration of MTDs with the rest of the trust components through the dedicated broker is provided in Sect. 6.

6 TRUST MESSAGE BROKER & TRUST COMPONENTS' INTEGRATION

A key characteristic that maximises the efficacy of IntellIoT's Trust Enablers is their tight integration and continuous communication through the dedicated Trust Broker, based on RabbitMQ[23] and using the AMPQ[24] binary protocol. More details on the different components and their role in this integration, along with their key interactions with other components are provided below.

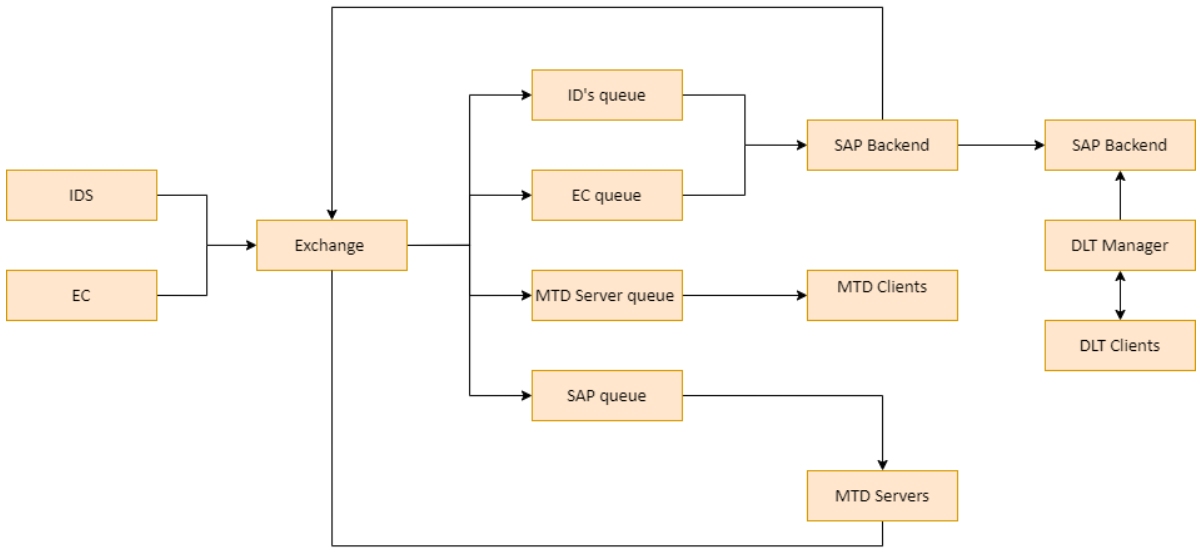
6.1 Trust Enablers' Integration Overview

The Trust IDS (see Sect. 4) continuously records events such as network traffic and analyses it for anomalies. They are responsible for aggregating pertinent evidence from multiple sources related to the operation of individual components, IDS computes a local trust value for each other node that it communicates with. When the trust value for a node drops below a threshold, it generates a warning. These warnings are left to be processed by the rest of the system in order to determine if an action is needed to be taken against offending nodes.

Similar to the IDS, the Event Captors (ECs; see Sect. 7.1) are software modules responsible for aggregating pertinent evidence from multiple sources related to the operation of individual components, as well as the overarching processes where these components are involved in, thus enabling the real-time, continuous assessment of the security posture of the IntellIoT system. ECs can be deployed across all layers of the IntellIoT architecture, from the robot arm (e.g., to monitor telemetry that may indicate abnormal activities) to device operating systems (e.g., to monitor running processes) and backend storage databases (e.g., to parse access logs or calculate uptime), transmitting their monitoring evidence to the Security Assurance Platform (SAP).

The SAP (see Sect. 2) is central to the trustworthiness components and has its own queue in the message broker pipeline. SAP digest messages respectively from the IDS queue and EC queue while in the same time through appropriate REST APIs it can communicate with the Distributed Ledger Trustworthiness (DLT) manager.

Finally, the MTDs (see Sect. 5) are responsible for mitigation actions once the SAP receives an Intrusion Warning from the IDS. MTDs are divided into two distinct components (the server MTD, and the client MTD) with the first having its own queue (MTD server queue) while MTD clients consumes messages from the latter queue.



[23] https://www.rabbitmq.com/
[24] https://www.rabbitmq.com/

Figure 29. Overview of Trust Enablers' integration via the shared Trust Broker

Each of the above-mentioned components are considered as producers and/or publishers to the Trust Broker, declaring its own exchange queue (Figure 29). There are in total 4 queues bounded to the topic exchange with 4 routing key patterns (#ids, #mtd, #sap, #ec, for Trust IDS, MTD, SAP and EC -related messages, respectively), while in the topic exchange the Queues are linked using the routing key patterns instead of a simple routing key.

The flow of a messages in Topic Exchange follows the typical publish/subscribe scheme. More specifically:

- A message Queue binds to an Exchange with a routing key pattern (P).
- A publisher sends a message with a routing key (K) to the Topic Exchange.
- The message is passed to the Queue if P matches with K. The routing key matching is decided as discussed below.
- The consumer subscribing to the Queue receives the message.

In terms of Topic Exchange requirements, the specific routing patterns are followed:

- A routing key in Topic Exchange must consist of Zero or more words delimited by dots
- A routing pattern is similar to a regular expression with only *, . and # allowed. The symbol star (*) means exactly one word allowed. Similarly, the symbol hash (#) means zero or more number of words allowed. The symbol dot (.) means – word delimiter.
- Multiple key terms are separated by the dot delimiter.
- If a routing pattern is ids.*, it means any message sent with the routing key "ids" as the first word will reach the queue. For example, ids.event will reach this Queue, but event.ids will not work.

6.2 IDS integration details

IDS exposes two topics, in JSON format, to the broker for the rest of the security components to consume and take appropriate actions. The first one is the trust topic `'ids.trust.[node id]'`, where each IDS instance publishes its trust values for the nodes it has communicated with. An example payload is:

```
{
  "10.0.0.1": 0.0,
  "10.0.0.2": 1.0,
              ...
}
```

The second topic is the `'ids.warn.[node id]'`, where each IDS instance publishes a list of nodes that are considered untrustworthy. An example payload is:

```
{
  "nodes": [
    "10.0.0.1"
  ]
}
```

6.3 MTD integration details

Apart from waiting input from the IDS topics described above, the following topics are used:

- *mtd.registration*
- *mtd.config.<client name>*
- *mtd.keepaliveReq.<client name>*
- *mtd.keepaliveResp.<client name>*
- *mtd.alert*
- *mtd.trigger*

Details on each of the four topics are provided in the subsections that follow.

6.3.1 MTD REGISTRATION TOPIC

MTD clients have write access to this topic and the server has read access. This is the topic where clients publish registration/deregistration requests.

The format is as follows:

```
type registrationCfg struct {
    Action   string
    NodeName string
    NodeIP   string
}
```

An example payload is provided below:

```
{
  "Action": "register",
  "NodeName": "TSI_sensor1",
  "NodeIp": "192.168.1.10"
}

{
  "Action": "deregister",
  "NodeName": "TSI_sensor1",
  "NodeIp": ""
}
```

6.3.2 MTD CONFIG TOPIC

A new subtopic is created for each registered client and each client has read access only to its own subtopic. The server has write access to all subtopics and sends personalised configurations to each client, possibly skipping clients that are deemed compromised.

The format is as follows:

```
type VPNConfig struct {
    //TUN
    CIDR string
    //UDP
    Port   int
    Protoc string
    //Cipher
    CipherKey  string
    CipherType cipher.CipherType
    //router
    ExtIP      string
    LocIP      string
    RoutesToAddmap[string]string
    RoutesToRemove []string
}
```

An example payload is provided below:

```
{
  "CIDR": "10.0.0.2/24",
  "Port": 30000,
  "Protoc": "udp4",
  "CipherKey": "3863766a4c4f553166502d716a324b69",
  "CipherType": 3,
  "ExtIP": "192.168.176.5",
  "LocIP": "10.0.0.2",
  "RoutesToAdd": {
"10.0.0.1": "192.168.176.4",
"10.0.0.2": "192.168.176.5",
  },
  "RoutesToRemove": [
"10.0.0.3"
  ]
}
```

6.3.3 MTD KEEP-ALIVE REQUEST AND KEEP-ALIVE RESPONSE TOPICS

Similarly to the configuration topic, a new subtopic is created for each registered client and each client has read access only to its own request subtopic and write access to its own response subtopic. The server has write access to all request subtopics and read access to all response subtopics. The server sends requests to which the clients must answer within a specific amount of time before being treated as disconnected. The payload is a randomly generated string that the client has to send back along with its status using its own response topic.

6.3.4 MTD AND SAP

The MTD client and server applications communicate over a secure channel provided by the Trust Message Broker. More specifically, the MTD Server consumes messages generated from IDS and takes mitigation actions when this is needed. When such an action is taken, the server needs to notify the Security Assurance Platform, as it is the component providing a holistic view of the current security and privacy posture of the system to the operators. This happens through the *'mtd.alert'* topic.

The reverse case is also needed. The Security Assurance Platform, through the event captors, can also identify attacks and trigger a mitigation action using the *'mtd.trigger'* topic.

Both of these follow the same payload logic as the *'ids.warn' topic:*

```
{
  "nodes": [
    "10.0.0.1"
  ]
}
```

6.4 Integration & interplay of Security Assurance Platform with the DLTs

Distributed ledger technologies (DLTs) are positioned as a key enabler for trusted and reliable distributed monitoring systems since these support immutable and transparent information sharing among involved untrusted parties. In DLTs, the authentication process relies on consensus among multiple DLT managers in the network. While the terms DLT and Blockchain will be used interchangeably, Blockchains are a type of DLT, where chains of blocks are made up of digital pieces of information called transactions and every node maintains a copy of the ledger. Therefore, in a DLT-enabled network, transactions contain, for example, monitoring control messages, and these are recorded and synchronized in a distributed manner in all the participants of the system. These participants are called miners or peers, and, in some specific DLTs, users are charged a transaction fee to perform (crypto) transactions. In addition, DLTs allow the storage of all transactions into immutable records and every record distributed across many participants. Thus, security in DLTs comes from the distributed characteristic, but also the use of strong public-key cryptography and strong cryptographic hashes.

The benefits of the integration of DLTs into IoT monitoring systems include:

i) guarantee of immutability and transparency for report data;
ii) removal of the need for third parties;
iii) development of a transparent system for heterogeneous secured monitoring networks to prevent tampering and injection of fake data from the stakeholders

While DLTs are a key trust enabler in the project, the bulk of the efforts on the development of the resource-aware DLT components developed within IntellIoT take place within Task 3.4 ("Decentralized trust via secure interaction & contracts"), and for Cycle 1 are documented in the corresponding deliverable, i.e., D3.4 – "Decentralized trust via secure interaction and contracts (first version)". For the sake of brevity, these details will not be repeated here.

Nevertheless, details on the integration of the DLT with the SAP will be provided below. This integration facilitates the trust-by-design approach of IntellIoT, as it allows the recording of security and privacy -pertinent evidence from the SAP in the Ledger, in a tamper-proof manner. This provides an additional layer of trust for said evidence aggregated at the SAP, which may include:

1) Evidence generated internally at the SAP (e.g., vulnerability or dynamic testing assessment results);
2) Evidence the SAP collects from monitoring and interacting with other Trust Enablers (e.g., Trust IDS alerts, or triggered MTD strategies)
3) Monitoring Evidence from monitoring the Event Captors deployed across the various layers of IntellIoT and the protected deployment.

Through interaction between the SAP and the DLT Manager, the above types of Evidence are recorded in the Ledger in an automated manner, and the entries (more specifically, transaction and block IDs) are returned to the SAP, to be provided to the SAP operator for verification (e.g., in the case of any audit).

More details on how each of the above three cases are relayed from the SAP to the DLT Manager are provided in the following subsections.

6.4.1 SAP TO DLT RECORD OF INTERNALLY GENERATED EVIDENCE

Messages of this type (e.g., assessment results) can be sent to the DLT for recording using a JSON format having the following structure:

```
"@timestamp": <timestamp>,
"event": {
"kind": <kind of event>,
"category": <category of event>,
"type": <type of event>,
"number_of_vulnerabilities": <the number of vulnerabilities>,
"critical_vulnerabilities": <the number of critical vulnerabilities>,
"high_vulnerabilities": <the number of high vulnerabilities>,
"medium_vulnerabilities": <the number of medium vulnerabilities>,
"low_vulnerabilities": <the number of low vulnerabilities>,

},
"src_ip": <dynamic tester ip>,
"src_port": <dynamic tester port>,
"assets":[  #< assets that were tested>
    "asset_1":{
    "asset_ip":<asset ip>,
    "asset_name":<asset name>,
    "asset_port":<asset port>,
    "port_protocol":<asset protocol>},
```

```
    ...
    "asset_N":{
    "asset_ip":<asset ip>,
    "asset_name":<asset name>,
    "asset_port":<asset port>,
    "port_protocol":<asset protocol>},
]
}
}
```

An example based on the above template is provided below:

```
{
"@timestamp": 2017-04-07T22:24:37.251547+0100,
"event": {
"kind": "alert",
"category": "dynamic_testing",
"type": "dynamic_testing_report",
"number_of_vulnerabilities": "{{X}}"
"critical_vulnerabilities": "{{Y}}"
"high_vulnerabilities": "{{Z}}"
"medium_vulnerabilities": "{{W}}"
"low_vulnerabilities": "{{V}}"
},
"src_ip": "192.168.122.149",
"src_port": 49324,
"assets":[
    "asset_1":{
    "asset_ip":"192.168.1.1"
    "asset_name":"test_asset"
    "asset_port":"443",
    "port_protocol":tcp},
    "asset_2":{
    "asset_ip":"192.168.1.2"
    "asset_name":"test_asset2"
    "asset_port":"443",
```

```
    "port_protocol":tcp},
]
}
```

Furthermore, a screenshot of such a message exchange between the SAP and the DLT Manager, from the early integration testing taking place between the two components, is shown in Figure 30.

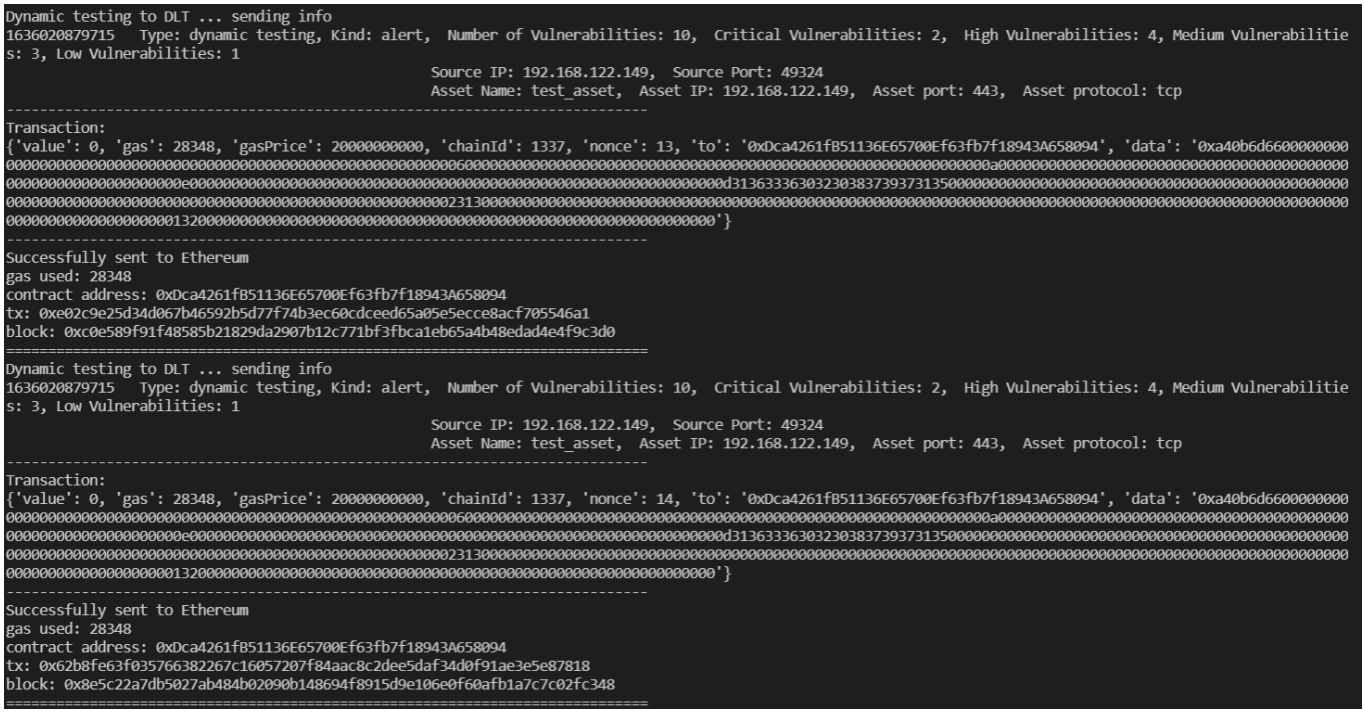


Figure 30. Recording of SAP Assessment Results in the Ledger, via interaction with DLT Manager (request & response).

6.4.2 SAP TO DLT RECORD OF TRUST ENABLER EVIDENCE

Messages of this type (e.g., alerts from the Trust IDS) can be sent to the DLT for recording using a JSON format having the following structure:

```
{
"@timestamp": <timestamp>,
"event": {
"kind": <kind of event>,
"category": <category of event>,
"type": <type of event>,
},
"src_ip": <source ip>,
"src_port": <source destination>,
```

```
"dest_ip": <destination ip>,
"dest_port": <destination port>,
"proto": <protocol>,
"app_proto": <protocol specs>,
}
```

An example based on the above template is provided below:

```
{
"@timestamp": 2017-04-07T22:24:37.251547+0100,
"event": {
"kind": "alert",
"category": "intrusion_detection",
"type": "anomaly",
},
"src_ip": "192.168.122.149",
"src_port": 49324,
"dest_ip": "69.195.71.174",
"dest_port": 443,
"proto": "TCP",
"app_proto": "tls",
}
```

A screenshot of such a message exchange between the SAP and the DLT Manager, whereby Trust IDS events aggregated by the former are related to the latter, is shown in Figure 31.



Figure 31. Recording of Trust IDS events in the Ledger, via interaction between SAP and DLT Manager (request & response)

6.4.3 SAP TO DLT RECORD OF MONITORING EVIDENCE

Messages of this type (e.g., Monitoring Rule triggers via Event Captor inputs) can be sent to the DLT for recording using a JSON format having the following structure:

```
{
"@timestamp": <timestamp>,
"event": {
"Profile": <the Assessment Profile name>,
"Asset Name": <The name of the Asset>,
"Source": <the source of the event, e.g. Filebeat, Auditbeat, Native>,
},
"Outcome": <The monitor result, Satisfaction/Violation>
}
```

An example based on the above template is provided below:

```
{
"@timestamp": 2017-04-07T22:24:37.251547+0100,
"event": {
"Profile": Availability,
"Asset Name": testasset,
```

```
"Source":  Native,
},
"Outcome": Satisfaction
}
```

A screenshot from such a message exchange, whereby a Monitoring Rule trigger is recorded from the SAP in the Ledger, is shown in Figure 32.

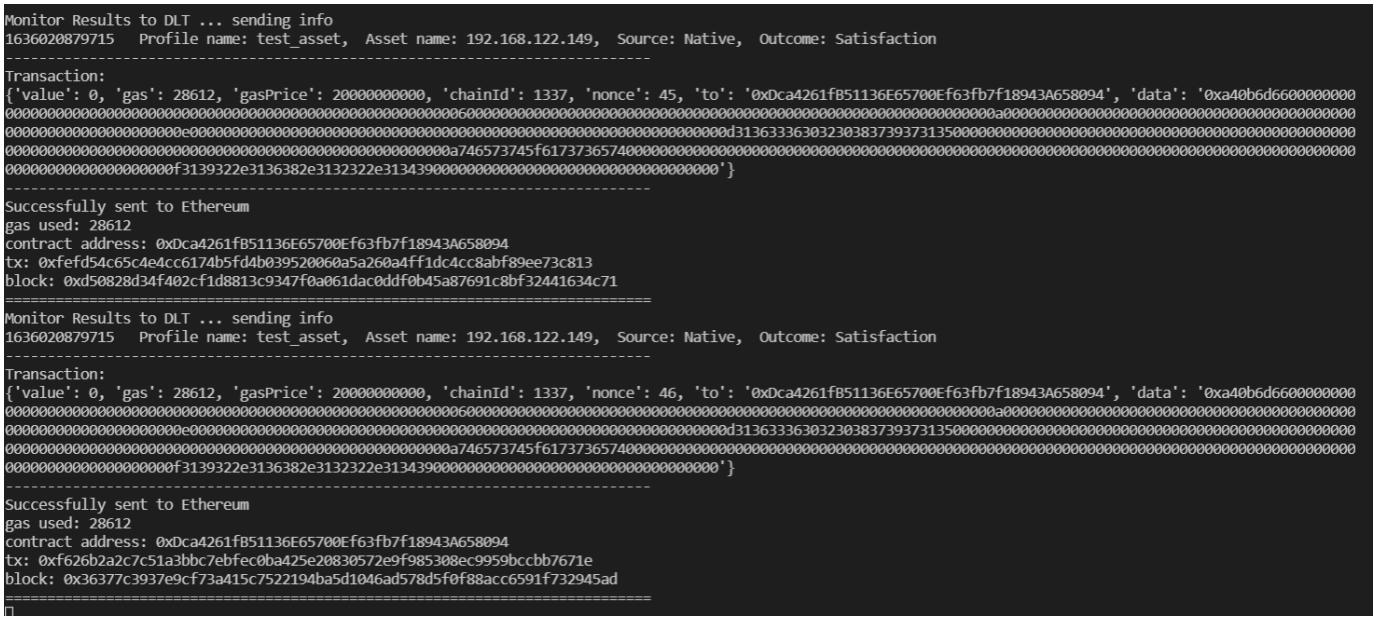


Figure 32. Recording of Monitoring Rule trigger evidence in the Ledger, via interaction between SAP and DLT Manager (request & response).

7 ADDITIONAL SECURITY, PRIVACY, AND TRUST PRIMITIVES

In addition to the various trust enablers defined above, IntellIoT also integrates a number of additional design choices and enablers that facilitate its trust-by-design approach. These are detailed in the subsections that follow.

7.1 Multi-layer Monitoring

Multi-layer monitoring is an important part of the trust-by-design in IntellIoT, as it provides the means to collect the necessary evidence needed to monitor and ensure that the related mechanisms work properly and safeguard the system and the users. This monitoring encompasses:

- the security posture of the IoT environment where IntellIoT is deployed, including the application assets and the IntellIoT components;
- the operation of all the other security, privacy, and trust primitives, as defined herein.

Evidence from the above sources leveraged in real-time to provide an up-to-date view of the security and privacy posture of the protected infrastructure and the IntellIoT framework, via the SAP component (see Sect. 2). More details in terms of the implementation are provided in the subsections that follow.

7.1.1 EVENT CAPTORS

The event captors are responsible for aggregating required evidence from multiple sources related to the operation of individual components, as well as the overarching processes where these components are involved in, enabling the real time, continuous assessment of the security posture of the IntellIoT system. Data and events are mostly collected through ElasticSearch based on lightweight shippers (referred to as "Beats", such as FileBeat[25], MetricBeat[26], and PacketBeat[27]), that forward and centralize log data. Data can also be collected through Logstash, an open server-side data processing pipeline that ingests data from a multitude of sources, transforms it, and then sends it to ElasticSearch.

The Elastic Common Schema (ECS) will be used as an open-source specification for defining a common set of fields among with the data model that will be used when storing event data in Elasticsearch. Furthermore, ECS will be used to specify the field names and datatypes to be stored while supporting uniform data modelling, enabling to analyse data from diverse sources.

7.1.2 EVENTS COMMON SCHEMA SPECIFICATION

At a high level, ECS provides fields to classify events and data in two different ways: "Where is the data is coming from?" and "What data is it?". ECS defines four categorization fields for this purpose[28]:

- *`Event.kind:`* Highest level in the ECS category hierarchy, providing high-level information about what type of information the event contains, without being specific to the contents of the event. Possible values include:
  - *`alert`*
  - *`event`*
  - *`metric`*
  - *`state`*
  - *`pipeline_error - signal`*

[25] https://www.elastic.co/beats/filebeat
[26] https://www.elastic.co/beats/metricbeat
[27] https://www.elastic.co/beats/packetbeat
[28] https://www.elastic.co/guide/en/ecs/current/ecs-event.html

- *Event.category:* Second level in the ECS category hierarchy, representing "big buckets" of ECS categories. Possible values include:
  - *authentication*
  - *configuration*
  - *database*
  - *driver*
  - *file*
  - *host*
  - *iam*
  - *intrusion_detection*
  - *malware*
  - *network*
  - *package*
  - *process*
  - *registry*
  - *session*
  - *web*
- *Event.type:* Third level in the ECS category hierarchy, representing a "sub-bucket" categorisation. When used along with "Event.category", it enables filtering events to a level appropriate for single visualization. Possible values include:
  - *access*
  - *admin*
  - *allowed*
  - *change*
  - *connection*
  - *creation*
  - *deletion*
  - *denied*
  - *end*
  - *error*
  - *group*
  - *info*
  - *installation*
  - *protocol*
  - *start*
  - *user*
- *Event.outcome:* Fourth level in the ECS category hierarchy. Denotes whether the event represents a success or a failure from the perspective of the entity that produced the event. Possible values include:
  - *failure*
  - *success*
  - *unknown*

Finally, the payload specification is as follows:

```
{
        "@timestamp": "<the time stamp>",
        "event": {
```

```
                "kind": "<>",
                "category": "<>",
                "type": "<>",
                "outcome": "<>",
        },
        "<payload>" : "all possible recorded from Elasticsearch (ECS fields) except
"event" gategories"
}
```

7.2 5G Security Considerations

To protect the communication infrastructure itself, IntellIoT will adopt the best practices also from the perspective of the 5G communications. In the following section, the 5G security architecture will be briefly described. A longer description is provided in the 3GPP TS 33.501 document.

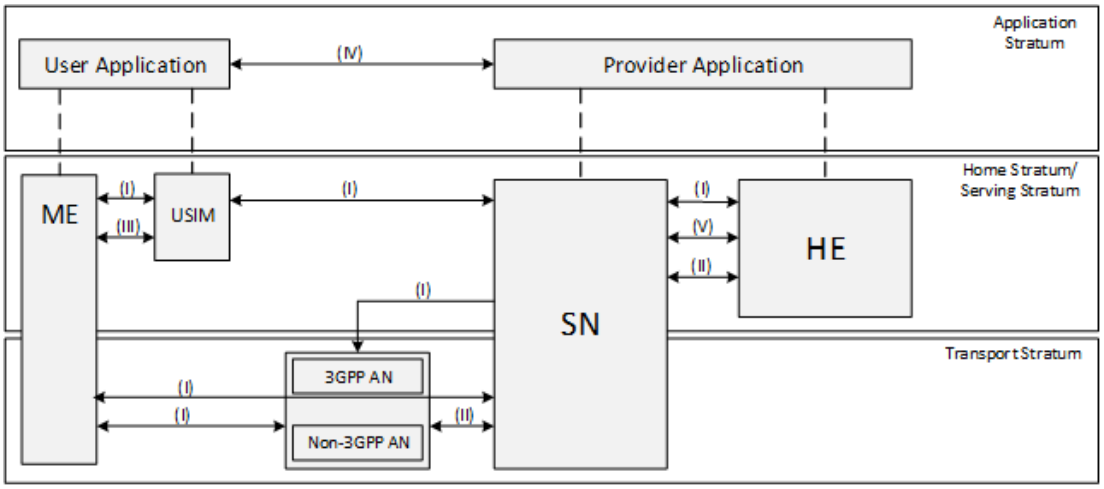


Figure 33. Generic 5G Security Architecture (source: 3GPP)

The generic 5G security architecture is depicted in Figure 33, which describes several 5G security domains:

- **Network access security (I):** the set of security features that enable a UE to authenticate and access services via the network securely, including the 3GPP access and non-3GPP access, and in particularly, to protect against attacks on the (radio) interfaces. In addition, it includes the security context delivery from the Serving Network (SN) to the Access Network (AN) for the access security.
- **Network domain security (II):** the set of security features that enable network nodes to securely exchange signalling data and user plane data.
- **User domain security (III):** the set of security features that secure the user access to mobile equipment.
- **Application domain security (IV):** the set of security features that enable applications in the user domain and in the provider domain to exchange messages securely. Application domain security is out of scope of the present document.
- **Service-based Architecture (SBA) domain security (V):** the set of security features that enables network functions of the SBA architecture to securely communicate within the serving network (SN) domain and with other network domains. Such features include network function registration, discovery, and authorization

security aspects, as well as the protection for the service-based interfaces. SBA domain security is a new security feature compared to the 4G security architecture as described in 3GPP TS 33.401.

- **Visibility and configurability of security (VI):** the set of features that enable the user to be informed whether a security feature is in operation or not (not displayed on Figure 34)

7.2.1 OVERVIEW OF COMMON 4G/5G SECURITY

The generic 3GPP security functions can be split in two categories:

- **Access-layer security –** this aims at securing the wireless link between the mobile terminal and the base station (4G eNB or 5G gNB) again security attacks. This is primarily done through mutual authentication between the UE and the gNB (gNB authenticating to the UE as well as the UE authenticating to the gNB) using cryptographic mechanisms stored in the USIM. Although the mechanisms have been generalized to support additional IETF-based security mechanisms, the Access-layer security between 4G or 5G network did not evolve significantly.
- **Network-domain security –** this aims at securing the various back-end entities in a 4G or 5G network. As depicted on Figure 35, these security mechanisms have been primarily designed in the case of roaming or virtual networks, as it remains largely assumed that network-domain entities operate in a monolithic and secured environment (i.e., owned by the same operator at the operator's premise). Accordingly, a 4G eNB or a 5G gNB should mutually authenticate with the visited PLMN (Public Land Mobile Network). Similarly, the visited PLMN must mutually authenticate to the Home HSS (the home database containing the access rights of the user).

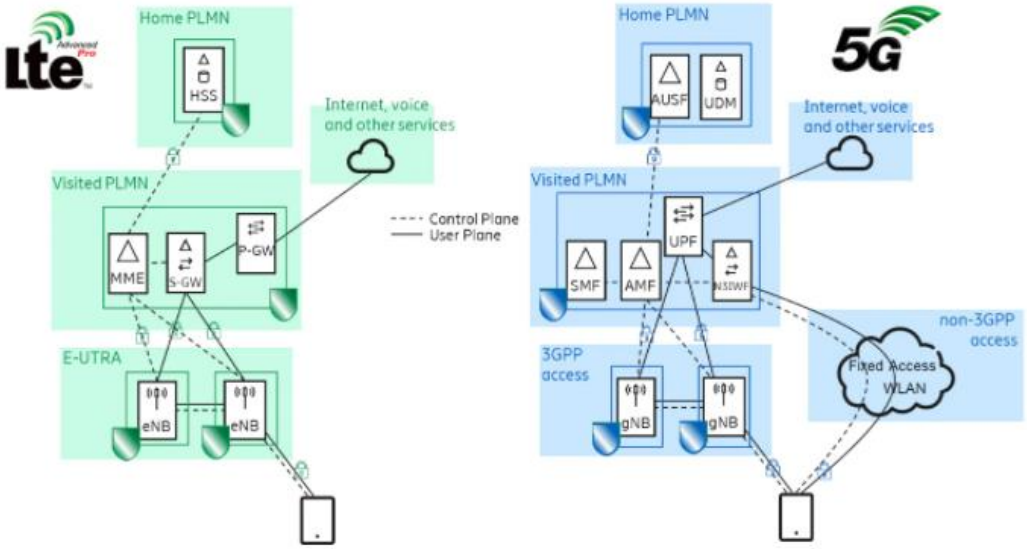


Figure 35. Network-domain security mechanisms (Source: Ericsson)

7.2.2 NEW 5G-SPECIFIC INNOVATIONS

If the 5G security architecture inherits many functions from previous generations (LTE or UMTS), it has a few major new security functions primarily adapted to handle three key 5G innovations: 5G end-to-end security, service-based architecture (i.e., slicing) and security on the user plane.

7.2.2.1 NEW AUTHENTICATION FRAMEWORK

The main authentication framework is known as the primary authentication and consists of the exchange of session keys between the UE and the eNB/gNB during the device registration phase. One limitation is that the security mechanisms are specific to 3GPP and cannot be extended beyond cellular operators, e.g., on a data network or a public internet. The new authentication framework enhances the primary authentication by supporting additional security and authentication algorithms as defined by the IETF under the name Extensible Authentication Protocols (EAP). EAP enables to use additional security credentials than those stored on the USIM card and would enable end-to-end security beyond the scope of cellular operators.

7.2.2.2 ENHANCED SUBSCRIBER PRIVACY

One major flaw in subscriber privacy in cellular networks is known as 'fake base stations', which can be applied through mechanisms known as IMSI catchers. Since 3G, a dual authentication has been enforced in order for the base station to authenticate to the user in the same way as the user needs to authenticate to the base station. Yet, side information, such as device capabilities or correlation between protocol messages could still reveal much of the identity of a subscriber. 5G aims at mitigating such risk by proactively limiting the number of messages sent in cleartext even at an early stage of protocol negotiation.

7.2.2.3 SERVICE BASED ARCHITECTURE AND INTERCONNECT SECURITY

Service-based architectures (SBA) is a major paradigm shift introduced in 5G networks. It relates to extend of traditional point-to-point interfaces between network functions to new service-based interfaces, where the overall network architecture is designed as a dynamic, on-demand exposed service set to network entities (e.g., a MEC service might or might not be exposed to a core network or a user-plane function (UPF). SBA additionally increases the level of security protocols from a purely L3 level (IPSec) to L4 TLS mechanisms to protect the integrity at transport level or even OAuth 2.0 protocols to secure the integrity at the application level.

7.2.2.4 INTEGRITY PROTECTION OF THE USER PLANE

If various mechanisms have been designed in 3G and 4G to protect the control plane, the user plane has always been a weak link of a cellular network. For example, if a mutual authentication has been introduced as early as 3G in order to guarantee that both a UE and a eNB/gNB are legitimate and allowed to use a particular service, data transmitted on 3G and 4G networks were still cleartext, unless covered by an additional end-2-end security level between a provider and consumer beyond the cellular infrastructure. 5G improves this by enforcing the *availability* of encryption and integrity protection mechanisms on the user plane between a UE and a gNB. By 'availability', it means that 5G operators must have such functions available if the devices require them, but they do not need to use it by default. Cryptographic mechanisms are indeed resource demanding and not all classes of mobile entities (ME) can support them.

7.3 Other best practices, processes, and technologies

In addition to all the above provisions towards a trustworthy-by-design framework, IntellIoT also adopts best practices and industry-established standards in all types of interactions that involve the processing, storage and transmission of data. Some key examples are provided below, while additional details will be provided in the deliverables documenting the setup of the demonstrator environments (i.e., D5.2 and D5.5), as some of these aspects are merely configuration provisions and not the development of novel technologies or specific building blocks.

Concerning web and other user interactions foreseen in IntellIoT (e.g., to define end user goals, or to access the SAP web front end) applications will use the Transport Layer Security (TLS) protocol (at v1.3 since August 2018 [6]). A successor to the Secure Sockets Layer (SSL) protocol, TLS uses a combination of symmetric and public-key encryption to achieve its goal. Both parties create a secure communication channel at the beginning using public-key cryptography. During this period, they can exchange all the parameters for their communication as well as agree on a symmetric key for the rest of the communication which is a less resource intensive type of communication.

Another important consideration in this regard is the protection of the Trust Broker that all Trust Enablers use to integrate and interact with each other (see Sect. 6). As mentioned above, this is based on the AMQP protocol. This protocol guarantees the immutability of the messages exchanged using hashes. On top of that the AMQPS variant is

going to be used, which is AMQP over TLS so no third party can have access to the information exchanged between the secure components.

Furthermore, for data at rest, meaning any data store (e.g., files or databases, such as the patient databased in Use Case 2) that exists in a node and needs to be accessed only by specific users/applications, appropriate encryption mechanisms will be used, appropriate to the sensitivity and criticality of the data to be protected. The encryption mechanisms will be combined with the authentication and authorization provided by the AAA component (see Sect. 3).

Finally, during the setup of the testing environment and, secure-by-default and fail-secure device configuration will be adopted for all key components (e.g., from network controllers and switches to sensor nodes and edge devices – tractor, robotic arm, etc.), to ensure that devices are secure out of the box (integrating all security-related bootstrapping procedures as needed) and, in the case of any failure, either by a fault or a malicious action, they will revert to a safe state.

8 CONCLUSIONS

This deliverable is the first output of Task 4.4 ("Trustworthy infrastructure by design") and, as such, documented the design and development of the first version of the security, privacy and trust enablers comprising the "Trust pillar" of IntellIoT, as defined within the corresponding first version of the framework's Architecture (as documented in D2.3).

As with all IntellIoT building block implementation tasks in WP3 and WP4, the components' specifications and associated interactions and requirements defined within D2.3, drove the design and development of the trust enablers presented herein. These covered all three pillars comprising the trust-by-design approach of IntellIoT, including continuous security assurance, security, privacy and trust primitives, as well as the multi-layer security monitoring.

In terms of next steps, the building blocks defined herein will be provided as input to Task 5.1, towards the delivery of the first (Cycle 1) integrated version of IntellIoT, to be documented in D5.1 – "Integrated IntellIoT framework & use case implementations (first version)".

Furthermore, the associated demonstration (in Task 5.2, documented in D5.2) and evaluation (in Task 5.3, documented in D5.3) that will follow will inform the Cycle 2 efforts of the project. Starting with updates to Task 2.1 (use case updates, to be documented in D2.4), Task 2.2 (requirements' updates, to be documented in D2.5), and Task 2.3 (architecture update, with final architecture documented in D2.6), these efforts will drive updates to all the trust enablers defined herein.

The Cycle 2 (i.e., final) versions of the trust enablers to be developed within Task 4.4, will include a complete feature set, including capabilities already planned for Cycle 2, but also integrating refinements identified through the Cycle 1 evaluation mentioned above. These will be documented in deliverable D4.8 – "Trust mechanisms (final version)", which is due in M30 of the project (i.e., March 2023).

REFERENCES

[1] Shanahan, M. (1999). The event calculus explained. In Artificial intelligence today (pp. 409-430). Springer, Berlin, Heidelberg.

[2] Internet Engineering Task Force (IETF), D. Hardt, RFC 6749, The OAuth 2.0 Authorization Framework, https://datatracker.ietf.org/doc/html/rfc6749

[3] Internet Engineering Task Force (IETF), M. Jones, J. Bradley, N. Sakimura, RFC 7519, JSON Web Token (JWT), https://datatracker.ietf.org/doc/html/rfc7519

[4] Internet Engineering Task Force (IETF), K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, G. Zorn, RFC 2637, Point-to-Point Tunneling Protocol (PPTP), https://datatracker.ietf.org/doc/html/rfc2637

[5] Internet Engineering Task Force (IETF), S. Frankel, S. Krishnan, RFC 6071, IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap, https://datatracker.ietf.org/doc/html/rfc6071

[6] Internet Engineering Task Force (IETF), E. Rescorla, RFC 8446, The Transport Layer Security (TLS) Protocol Version 1.3, https://datatracker.ietf.org/doc/html/rfc8446

[7] G. Hatzivasilis, I. Papaefstathiou and C. Manifavas, "SCOTRES: Secure Routing for IoT and CPS," in *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 2129-2141, Dec. 2017, doi: 10.1109/JIOT.2017.2752801.

[8] Internet Engineering Task Force (IETF), J. Schaad, RFC 3565, Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS), https://datatracker.ietf.org/doc/html/rfc3565

[9] NIST, FIPS PUB 197, "Advanced Encryption Standard (AES)," Nov. 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197

ANNEX A – INTELLIOT ASSET MODEL SPECIFICATION FORM

This Annex provides several screenshots of the Asset Model specification form that was used to define the IntelIIoT asset model, covering the general tab (Figure 36), the tab for defining software assets (Figure 37), as well as the ones for data (Figure 38) and hardware assets (Figure 39).

Please make sure that before submitting the excel, only rows with actual data must be filled. To check if empty rows exist, please navigate to "Find & Select" -> "Go to special..." -> Check the "Blanks" box and delete (right click, "Clear Contents" all the blank rows)

Software Asset (Attributes)

Name (*)	Vendor (*)	Version (*)	Status(*)	Type (*)	Kind	Value	Currency	ActiveTo	Description	Contains (0..*)	Controls(0..*)	Processes(0..*)	Stores (0..*)	Transmits (0..*)	DeployedOn (0..*)	Supports (0..*)
Name of the software asset (e.g. Tomcat)	Vendor of the asset (e.g. Apache)	Version of the asset (e.g. 1.3.1)	Draft or Final. If draft, the asset will not take part in the assessment.	SAL[1] or PAL[2]	Service or Component	Monetary value of the asset - if not applicable insert 0	Currency of the value (e.g. EUR)	Timestamp that provides information of the date that asset will cease to exist	Brief description of the asset	A software Asset may contain 0..* **software or data** assets	A software asset may control **0..* software,hardware data assets and processes**	A software asset can process 0..* **data** assets	A software asset can store 0..* **data** assets	A software asset can transmit 0..* **data** assets.	A software asset can be deployed on 0..* **hardware** assets	A software asset can support 0..* processes

[1] An application layer software module (the source code of a software implemented within the organization etc, Apache Hadoop, MySQL etc.)

[2] An abstract software platform (i.e., a database connector, the JVM, a web server, OpenStack etc).

Hardware Asset (Attributes)

Name (*)	Vendor (*)	Version (*)	Category(*)	Status(*)	Value	Currency	ActiveTo	Description	Contains (0..*)	Controls(0..*)	Processes(0..*)	Stores (0..*)	Transmits (0..*)	Deploys (0..*)		
Name of the hardware asset (e.g. Zbook)	Vendor of the asset (e.g. HP)	Version of the asset (e.g. 15R)	Category of Hardware (compute or network)	Draft or Final. If draft, the asset will not take part in the assessment.	Monetary value of the asset - if not applicable insert 0	Currency of the value (e.g. EUR)	Timestamp that provides information of the date that asset will cease to exist	Brief description of the asset	A hardware Asset may contain 0..* **software or hardware** assets	A software asset may control **0..* software and hardware assets**	A hardware asset can process 0..* **data** assets	A hardware asset can store 0..* **data** assets	A hardware asset can transmit 0..* **data** assets.	A hardware asset deploys 0..* **software assets**		

Hardware Modules

Port Module (0..*)	CPU Module (1..*)	Memory Module (0..*)	Network Adapter (1..*)	Disk Module (0..*)	Power Supply Module (0..*)	Motherboard Module (0..*)
The IO Ports of the hardware asset	The CPU(s) of the hardware asset	The memory(ies) modules of the hardware asset	The network adapter(s) of the hardware asset	The disk modeuls of the hardware asset	The power supply(ies) modules of the hardware assets	The motherboard(s) modules of the hardware assets

Person Asset (Attributes)

First Name (*)	Last Name (*)	Email (*)	Value (*)	Currency(*)	ActiveTo	Description	Role (1..*)	Controls(0..*)	InvolvedIn (0..*)
First Name of the person	Last name of the person	Email	Monetary value (can be an estimation of the salary)	Currency of the value (e.g. EUR)	Timestamp that provides information of the date that asset will cease to exist	Brief description of the person	Role within the organisation	A person asset may control **0..* assets (software,hardware,data,processes and person)**	A person asset may be involded in **0..* processes**

Data Asset (Attributes)

Name(*)	Category(*)	State(*)	Value (*)	Currency(*)	ActiveTo	Description	GDPR Data Processor (0..*	Contains (0..*)	Controls(0..*)	ProcessedBy(0..*)	StoredBy (0..*)	TransmittedBy (0..*)
Data Name	Category of data	State of data	Monetary Value	Currency of the value (e.g. EUR)	Timestamp that provides information of the date that asset will cease to exist	Brief description of the data	Person asset that has a	A data asset can contain 0..* data assets	A data asset can control 0..* data assets	A data asset can be processed by 0..* assets (software,hardware, process,person)	A data asset can be stored by 0..* assets (software,hardware,process,person)	A data asset can be transmitted by 0..* assets (software,hardware,process,person)

Process (Attributes)

Name(*)	Description	ActiveTo	Contains (0..*)	Controls(0..*)	Involves(0..*)	Stores (0..*)	Transmits (0..*)	Process (0..*)	DependsOn (0..*)	SupportedBy(0..*)
Process Name	Brief description of the process	Timestamp that provides information of the date that asset will cease to exist	A process can contain 0..* assets (process,data,hardware ,software,person)	A process can control 0..* other processes	A process can involve 0..* **person** assets	A process can store 0..* **data** assets	A process can transmit 0..* **data** assets.	A **process** can process 0..* data assets	A process can depend on 0..* assets (process,data,hardware,software,person	A process can be supported by 0..* **software** assets

Note:

0..* — *Zero or more = The element may be repeated any times or be absent (optional)*

1..* — *One or more = The element may be repeated any times or at least one time (mandatory)*

General | Software Asset | Hardware Asset | Hardware Modules | Data Asset | Person Asset | Process Asset | Lists

Figure 36.General tab of the asset model specification form (overview & guidelines)

ID (Auto Increment)	Name (*)	Vendor (*)	Version (*)	Type (*)	Status (*)	Kind	Currency	Value	ActiveTo (Date)	Description	Contains (0..*)	Controls(0..*)	Processes(0..*)	Stores (0..*)	Transmits (0..*)	DeployedOn (0..*)	Supports (0..*)
1	MySql	Oracle	8.0.16	SAL	draft	Service	EUR			DBMS				15		12	
2	Rabbitmq	Pivotal Software	3.8.3	SAL	draft	Service	EUR			Message Broker					14	12	
3	Docker	Docker Inc.	19.03.6	SAL	draft	Service	EUR			Monitoring module	6,9					12	20
4	Mongodb	Mongodb	4.2.1	SAL	draft	Service	EUR			DBMS				16		12	
5	JDK	Oracle	1.8.0	PAL	draft	Service	EUR			Language Package						12,3	
6	Tomcat	Apache	9.0.27.0	SAL	draft	Service	EUR			Web Server	7	13				12	
7	Api Manager	Wso2	3.0.0	SAL	draft	Service	EUR			API Management			13			12	19
8	Spring Boot	Pivotal Software	2.2.4	SAL	draft	Service	EUR			Road event database REST API						3	
9	Phpmyadmin	Phpmyadmin	5.0.2	PAL	draft	Service	EUR			Language Package						12	7
10	Ubuntu	Ubuntu	18.04	PAL	draft	Service	EUR			Operational System	1,2,3,4,5,6,7,8,12,13					12	
11	Http server	Apache	2.4.29	SAL	draft	Service	EUR			Web server						12	

Figure 37. Software asset entries (indicative)

ID	Name(*)	Category(*)	State(*)	Status (*)	Value	Currency	ActiveTo (Date)	Description	Data Processor		Contains (0..*)	Controls(0..*)	ProcessedBy(0..*)	StoredBy (0..*)	TransmittedBy (0..*)
									Asset ID	Relation					
13	credentials	security	at_rest	draft		EUR		Sensitive data, used for the login in C4IIoT platform	7	Transmits					
14	assessment results	security	in_transit	draft		EUR		Security assessment results	2	Transmits					
15	assessment models	other	at_rest	draft		EUR		Models from database to the Docker asset	3	Processes					
16	events_toDB	operational	in_transit	draft		EUR		Events to Mongo DB	4	Stores					
17	events_from_Rabbit	operational	in_transit	draft		EUR		Events from Message broker that sent for analysis to the Docker monitor	3	Processes					

Figure 38. Data asset entries (indicative)

ID	Note: Insert the ID of each module - First Define the Module in the Hardware Modules Tabs															Hardware Module							
	Name (*)	Vendor (*)	Version (*)	Status (*)	Category(*)	Value	Currency	ActiveTo	Description	Contains (0..*)	Controls(0..*)	Processes(0..*)	Stores (0..*)	Transmits (0..*)	Deploys (0..*)	Port	Cpu	Memory	Disk	Network Adapter	Motherboard	Video	Power Supply
12	Assurance Tool VM	DELL	6800	final	Compute		EUR		Virtual machine provided for deploying Assurance platform tool							....	i7-8700 Box,GenuineIntel,8,4,2,4 GHz ,,,800000006,12,FALSE	VENGEANCE LPX,32,TRUE,DDR4 SDRAM,2400000000,Corsair,CORSAIR	Samsung 970 EVO,Samsung Electronics Co Ltd.100,,TRUE,970 Evo Plus,NVMe	00:0a:95:9d:68:16,Integrated,192.0.2.1,10.8.0.1,2001:0db8:85a3:0000:0000:8a2e:0370:7334, ,fe8,,,	GIGABYTE,Gigabyte - aorus pro,z390 ,,GIGABYTE software,F11,13/9/2019,TRUE	AMD,Radeon RX 5700,Navi 10,,1465/1725,8,GDDR6,,AMD,TRUE,FALSE	Enermax,Triathlor Eco etl550awt-M,650,,TRUE,FALSE,TRUE,TRUE,FALSE,TRUE,TRUE

Figure 39.Hardware asset entry (indicative)

ANNEX B – ASSET MODEL GRAMMAR

Examples of the use of the Asset Model Grammar used in the SAP are provided below (examples for Person, Software and Hardware assets, specifically).

Person(firstName("Demo"),lastName("User"),email("demo@sphynx.ch"),project("IntellIoT"),organisation("Sphynx Analytics Ltd."),roles(Project end-user))

SoftwareAsset(vendor("Oracle Corporation"),version("9.6"),name("MySQL Server"),kind(Service),type(SAL),project("IntellIoT"),organisation("Sphynx Analytics Ltd."),description("Relational Database"))

SoftwareAsset(vendor("Mongodb"),version("4.2.1"),name("Mongodb"),kind(Service),type(SAL),project("IntellIoT"),organisation("Sphynx Analytics Ltd."))

SoftwareAsset(vendor("Wso2"),version("3.0.0"),name("Api Manager"),kind(Service),type(PAL),project("IntellIoT"),organisation("Sphynx Analytics Ltd."),owner("staff"),description("API Manager")),

HardwareAsset(vendor("DELL"),version("6800"),name("Assurance ToolVM"), value(3000.0),currency(EUR),hwType(compute),project("IntellIoT"),organisation("Sphynx Analytics Ltd."),CpuModule(name("Intel i7-8700"),manufacturer("GenuinIntel"),numberOfCores(4),numberOfThreads(8),baseClock(2.4),cache(12),MemoryModule(name("VENGEANCE LPX"),size(32),type("DDR4"),speed(3200.0),manufacturer("Corsair")),NetworkAdapterModule(connectionType(Integrated),MAC("00:0a:95:9d:68:16"),NetworkConfiguration(ipv4("195.201.62.1"),Subnet Mask("255.255.0.0"))))