Title: *Trustworthiness mechanisms for Smart IoT Systems - First version*

Authors: *Anne Gallon (EVIDIAN), Christophe Guionneau (EVIDIAN), Valérie Clément (EVIDIAN), Samuel Mamuye (EVIDIAN), Jean-Christophe Durieu (EVIDIAN), Thierry Winter (EVIDIAN), Erkuden Rios (TECNALIA), Eider Iturbe (TECNALIA), Angel Rego (TECNALIA), Saturnino Martinez (TECNALIA), Borja Urquizu (TECNALIA), Hui Song (SINTEF), Rustem Dautov (SINTEF), Phu Nguyen (SINTEF), Brice Morin (SINTEF)*

Editor: *Anne Gallon (EVIDIAN)*

Reviewers: *Sergio Jiménez Gómez (INDRA), Jean-Yves Tigli (CNRS), Nicolas Ferry (SINTEF)*

Identifier: *Deliverable # D4.2*

Nature: *Other*

Date: *30 June 2019*

Status: *Final*

Diss. *level: Public*

Executive Summary

This deliverable provides the first version of the Security and Privacy Monitoring and Control Enabler and the Robustness and Resilience Enabler of the ENACT Trustworthiness toolkit. This includes the description of the prototypal developments made for the two main enablers: the Robustness and Resilience Enabler and the Security and Privacy Monitoring and Control Enabler. In addition, this document provides a status of the requirement coverage.

The research leading to these results has received funding from the European Community's H2020 Programme under grant agreement n° 780351 (ENACT).

Members of the ENACT consortium:

SINTEF AS	Norway
EVIDIAN SA	France
INDRA Sistemas SA	Spain
FundacionTecnalia Research & Innovation	Spain
TellU AS	Norway
Centre National de la Recherche Scientifique	France
Universitaet Duisburg-Essen	Germany
Istituto per Servizi di Ricovero e Assistenza agli Anziani	Italy
Baltic Open Solution Center	Latvia
Elektronikas un Datorzinatnu Instituts	Latvia
Montimage	France
Beawre	Spain

Revision history

Date	Version	Author	Comments
10-01-2019	V0.1	Erkuden Rios	TOC proposed
17-01-2019	V0.2	Erkuden Rios	TOC updated and agreed
05-03-2019	V0.3	Anne Gallon	
02-05-2019	V0.4	Erkuden Rios	TOC updated and agreed
06-05-2019	V0.4.1	Anne Gallon	§ 4.2.1 CAAC
23-05-2019	V0.5	Hui Song, Brice Morin	§3.2 and §4.2.3 Diversifier at design time and runtime
03-06-2019	V0.5	Phu Nguyen	§3.1.3
04-06-2019	V0.5.1	Hui Song, Rustem Dautov	Finalizing §3.2 and §4.2.3
06-06-2019	V0.5.2	Anne Gallon	§ 6 References
11-06-2019	V0.5.2.1	Erkuden Rios, Borja Urquizu, Angel Rego	§ 1.2, § 2, § 4.2.2, § 4.2.4
11-06-2019	V0.5.2.2	Eider Iturbe, Saturnino Martinez	§ 4.1
12-06-2019	V0.6	Anne Gallon	Version for internal review
20-06-2019	V0.6.1	Hui Song, Eider Iturbe, Erkuden Rios, Anne Gallon	Manage the remarks of the reviewers
24-06-2019	V0.7	Anne Gallon	Version for second internal review
28-06-2019	V1.0	Anne Gallon	Final version

Contents

1 Introduction

1.1 Context and objectives

The WP4 in ENACT aims at developing methods and tools supporting the security, privacy and resilience of Smart IoT Systems (SIS) throughout the DevOps process cycle (see Figure 1). Smart IoT Systems in ENACT are next generation IoT systems which need to perform distributed processing and coordinated behaviour across IoT, edge and cloud infrastructures, manage the closed loop from sensing to actuation, and cope with vast heterogeneity, scalability and dynamicity of IoT devices and their environments.

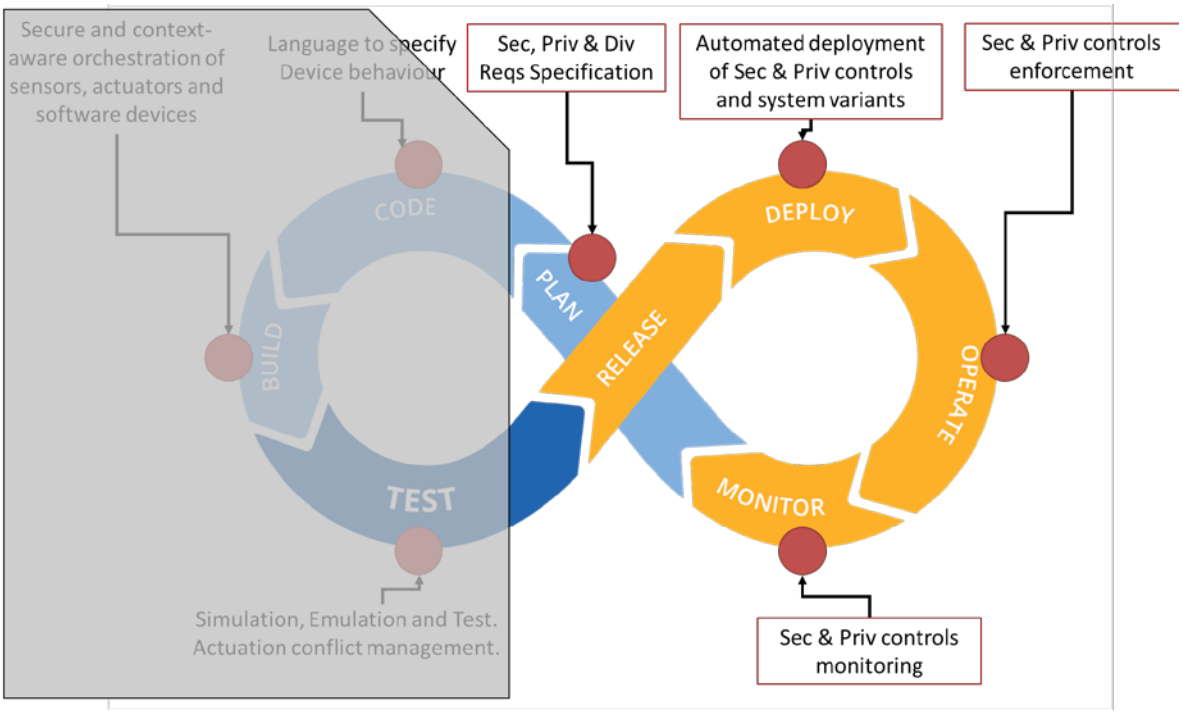

Figure 1. ENACT WP4 focus within DevOps cycle

As explained in D4.1, the WP4 deals with supporting **Security** (preservation of confidentiality, integrity and availability), **Privacy** and **Resilience of SIS**. In ENACT, SIS Resilience is based on enabling system components' software diversity to reduce the exposure of particular faults of the SIS to potential attackers as well as promptly recover from external perturbations.

The results of WP4 will be shaped as two main enablers:

- **Robustness and Resilience Enabler:** In order to contribute to smart IoT systems trustworthiness, this enabler will increase resilience of SIS by injecting or managing software diversity. This implies that each instance of a service has a different implementation and operates differently, still ensuring that its global behaviour is consistent and predictable. The enabler will automate the introduction and management of diversity in smart IoT systems, and in the following it will be referred to as *Code Diversifier* or *DivEnact*.
- **Security and Privacy Monitoring and Control Enabler:** This enabler includes mechanisms and tools to control the security and privacy behaviour of IoT systems and to early detect anomalies by continuously monitoring security metrics that will be defined during the project. This includes early reaction models and mechanisms that address adaptation and recovery of the IoT application operation in case of deviation from the expected behaviour of monitored security and privacy related metrics. Note that the security and privacy-aware behaviour of the

SIS complements other adaptation mechanisms to support SIS trustworthiness proposed by WP3 (see deliverable D3.2). Specific focus will be done on the confidentiality and integrity of data and services. The enabler will include a Context-Aware Access Control tool for advanced access control and authorization mechanisms tailored to smart IoT systems. Today, no protocol can deliver dynamic authorization based on context for both IT (information technologies) and OT (operational technologies) domains.

The deliverable D4.1 focused on the state-of-the-art and use case requirement analysis to derive the security, privacy and resilience mechanisms necessary to support security-by-design, privacy-by-design, resilience-by-design as well as monitoring and operational control of these aspects in SIS.

This D4.2 document is linked to the delivery of the prototypes of the tools provided through the WP4 enablers. It focuses on the use case requirement coverage and presents the operation of the different tools and how they can be used in the ENACT framework.

1.2 Achievements

The following table summarises the achievements of WP4 at the time of delivering D4.2.

Table 1. Achievements of ENACT WP4 at the time of D4.2 delivery.

Objectives	Achievements so far and future work
State-of-the-art on IoT security, privacy and resilience	In D4.1 we conducted an extensive analysis of the state-of-the-art on approaches for security, privacy and resilience of SIS. We specifically focused on four topics: 1. IoT Security and Privacy challenges. 2. IoT Security- and Privacy-by-design. 3. IoT Security and Privacy assurance. 4. Software diversity as resilience mechanism in SIS.
Security, and privacy-aware design and orchestration of IoT systems i) The languages and formalisms to enable the specification of the security and privacy requirements of smart IoT applications as part of the overall design, including the corresponding security and privacy metrics and probes allowing appropriate monitoring.	Description of possible mechanisms and procedures to express security and privacy requirements and controls in smart IoT systems design. Initial design and implementation of Security & Privacy monitoring enabler is ready. Initial formalisms to support to security and privacy specification at Orchestration GeneSIS model is ready.
ii) Risk model characterizing potential security and privacy risks, considering both the characteristics of infrastructure devices and requirements of the smart IoT application (this will be integrated with WP2 risk driven orchestration and	Addressed in D2.2

the decision support system for selection of the devices).	
iii) Metrics of software diversity of individual services and the whole system.	Initial diversity mechanisms already designed and first implementation ready as GeneSIS Orchestrator tool plugin.
Robustness, security and privacy enforcement in smart IoT systems	Initial design and implementation of context-based authentication and authorisation of devices and services. Specification of the interfaces of the Context-aware Access control tool, and description of the various ways this tool can be used to secure the IoT accesses, considering the context-awareness. The context-awareness aspect of these mechanisms is currently defined in a high-level specification. The next steps are to implement this aspect, and to check the tool in large-scale IoT systems. Initial design of diversity mechanisms to diversify IoT services, i.e., to automatically generate diverse versions of IoT services from the same ThingML model. In addition, initial research and prototyping is conducted on the diversity controlling at runtime, with a diversifier tool named DivEnact. Initial design and implementation of the IoT Platform level security and privacy mechanisms: security extensions of SMOOL (SOFIA platform). In the future initial enforcement mechanisms will be extended to address the adaptation and recovery of the IoT application operation in case of monitored metrics deviation from the normal (risk under control) behaviour.
Security and privacy monitoring of smart IoT systems	Initial design and implementation of the mechanisms and prototypes for continuously monitoring the security and privacy behaviour of IoT application and early detect anomalies have been developed. The monitoring will capture data from different layers: network, system and application in order to enrich the detection and increase its accuracy.

1.3 Structure of the document

After the introductory section, the reminder of the document is structured as follows.

Section 2 analyses the coverage of the requirements of ENACT use cases with respect to SIS security, privacy and resilience, to demonstrate the degree of fulfilment by initial WP4 prototypes of those requirements.

Section 3 describes the different methods and mechanisms being offered as part of the ENACT solution to support SIS developers in creating trustworthy SIS, by providing security-by-design, privacy-by-design and resilience-by-design techniques to be adopted in SIS development.

Section 4 describes the different mechanisms being developed as part of the ENACT solution to support IoT system operation with respect to ensuring a secure, resilient and privacy-respectful behaviour.

1.4 Acronyms and abbreviations

API	Application Programming Interface	KP	Knowledge Processor
BTE	Binary Tree Encryption	OIDC	OpenID Connect
CAAC	Context-aware Access Control	OS	Operating System
GDPR	EU General Data Protection Regulation	OT	Operational Technologies
GUI	Graphical User Interface	PDP	Policy Decision Point
GW	Gateway	REST	Representational State Transfer
HTTP	HyperText Transfer Protocol	S&P	Security & Privacy
HTTPS	HyperText Transfer Protocol Secure	SIS	Smart IoT System
IDS	Intrusion Detection System	SSAP	Source Service Access Point
IoT	Internet of Things	SSH	Secure SHell
IP	Internet Protocol	TCP	Transmission Control Protocol
IPS	Intrusion Prevention System	TLS	Transport Layer Security
IT	Information Technologies	XACML	eXtensible Access Control Markup Language
ITS	Intelligent Transportation System	WAM	Web Access Manager
JWT	JSON Web Token		

2 Analysis of Use cases requirements coverage

The analysis of use cases carried out within WP1 produced a number of usage scenarios from where the requirements for the ENACT enablers were derived. Such analysis was complemented in WP4 with a dedicated questionnaire that inquired about different security and privacy aspects that clarified the needs of the use cases in terms of (personal) data protection both at rest and in transfer (see deliverable D4.1).

Together with the initial analysis of security and privacy aspects, the use cases identified a number of requirements related to the trustworthiness support in ENACT (collected in D1.1). From them, the requirements related to security, privacy and resilience will be used to evaluate the success of WP4 solutions. The next table summarizes the degree of fulfilment by initial WP4 prototypes of those requirements. In particular, the considered requirements refer to the following tools: (i) context aware access control, (ii) software diversity of IoT systems, and (iii) privacy and security monitoring and control.

As it can be seen, in the eyes of the end-users in ENACT, all requirements are high or medium priority and all medium priority ones are recommended or nice to have features. These features will be addressed during the project lifetime though the focus of WP4 work will be on addressing high and mandatory requirements.

In the last column of the table, Coverage, we have indicated the degree of fulfilment of the requirement by the initial version of the corresponding WP4 Enabler. The future D4.3 will report on the status of the requirements coverage by the final implementation of WP4 Enablers.

It is worth noting that in some cases, the requirement is very specific to the particular use case IoT application and ENACT will not address it by means of an external enabler, but rather rely on the particular IoT application itself to resolve it.

However, for most requirements the software components or modules that will be developed in WP4 (Code Diversifier, Security and privacy Monitoring Enabler, Context Aware Access Control Enabler, etc.) will address the issue. See section 3 and 4 to understand how each of the modules fits into ENACT Enablers.

Table 2. Security, Privacy and Resilience Requirements Coverage by initial WP4 Enablers.

ID	Statement	Source[1]	Brief description	Priority[2]	Need[3]	How to Address in WP4/IoT app	Coverage
DO-4	***4. ENACT Trustworthiness Toolkit***						
DO-4.1	***4.1. Robustness & Resilience Enabler***						
DO-4.1.1	Gateway recovery and factory reset	2	There is a need to allow for resetting the Medical Gateway to factory default when something goes wrong, and then get the GW operational after reset	H	M	Diversifier	**(Partly covered)** DivEnact covers the bootstrap of gateway, which works both from scratch and as a recovery from factory reset. The actual reset currently requires device-specific mechanisms and end-user intervention. This will be the next step.

[1] Source - 1: ITS use case, 2: Digital Health use case; 3: Smart Building use case.

[2] Priority - H: High; M: Medium, L: Low.

[3] Need - M: Mandatory; R: Recommendation.

DO-4.1.2	Handle Medical Gateway failure situations	2	The Medical Gateway should quickly recover from being unresponsive. In addition to extensive and continuous testing this, includes features for handling the failure for example through remote access in a safe mode.	M	R	Diversifier	**(Partly covered)** DivEnact deploys default applications into gateways automatically and remotely. The next step is to investigate and prepare a minimal "safe mode" deployment as a default deployment for the purpose of trouble shooting.
DO-4.1.3	Roll back configuration	2	In case a deployment of a new configuration fails. The GW should be able to rollback to the previous configuration and notify the Operator	H	M	Diversifier	**(Covered)** Rollback is supported by DivEnact, in particular, it records the last deployment that was used successfully, and when rollback is required for a device, DivEnact pushes the recorded deployment to this specific device.
DO-4.2.x	***4.2. Risk-Driven Decision Support Enabler***					Addressed by WP2.	
DO-4.3	***4.3. Security and Privacy Monitoring and Control Enabler***						
DO-4.3.1	Authentication	1	Authentication procedures are applied to treat every data packet.	H	M	IoT app[4]	Not to be addressed by WP4.
DO-4.3.2	Authentication invalid	1	A procedure to deal with invalid authentication of the elements and users must be designed.	H	M	IoT app	Not to be addressed by WP4.
DO-4.3.3	Security not variable	1	The security measurements are not adapted if the system is running	H	M	S&P[5] Monitoring Enabler	**(Covered)** The settings of the monitoring metrics will be set before the monitoring starts.
DO-4.3.4	Authentication levels	1	Several authentication levels would be designed.	M	R	IoT app	Not to be addressed by WP4.
DO-4.3.5	Attacks historical	1	An historical of that would be created.	M	R	S&P Monitoring Enabler	**(Covered)** The prototype stores statistics and information on anomalies detected.
DO-4.3.6	Things and On Board GWs identification management	1	The Ids of the system elements must be checked.	H	M	IoT app	Not to be addressed by WP4.
DO-4.3.7	Access security	1	The users must be authorized to access to the tool which manages the SW updates.	H	M	IoT app	Not to be addressed by WP4.
DO-4.3.8	Orchestration Interface	1	The Monitoring enabler awares the Orchestration of alerts related with a shift in some of the elements performance after processing the data gathered.	H	M	S&P Monitoring Enabler	**(Partly covered)** The prototype is able to detect whether deployed security mechanisms are working properly.

[4] IoT app – IoT application (particular to the use case).

[5] S&P – Security & Privacy.

DO-4.3.10	Alarm thresholds configuration	3	The Trustworthiness Monitoring enabler should enable the user to set the desired thresholds to raise cybersecurity alarms.	H	M	S&P Monitoring Enabler	**(Partly covered)** The prototype will allow to set desired thresholds on the metrics to raise alerts, though this functionality is not yet ready.
DO-4.3.11	Security enforcement	3	The Trustworthiness Monitoring enabler should work together with Trustworthiness Adaptation Enabler which helps reacting to attacks or incidents	H	M	S&P Control Enabler	**(Partly covered)** Security Adaptation Enabler is now the Control Manager within the S&P Monitoring and Control Enabler. Some reactions will be automated (e.g. at IoT Platform level) and some will rely on humans (via the DevOps cycle).
DO-4.3.12	Protection of person sensitive data	2	Secure data management across IoT edge and cloud is severe as the system typically handle person sensitive data.	H	M	CAAC[6], S&P Control Enabler	**(Covered)** The SMOOL IoT Platform has been extended to support encryption and other security controls. The Context-Aware Access Control tool provides access control for both users and devices.
DO-4.3.13	Monitoring and control	2	There is a need to do Real-time monitoring of a set of Medical Gateways and to receive proper notifications with useful information in case of errors.	H	M	S&P Monitoring Enabler	**(Partly covered)** The notification functionality is still being developed in the S&P Monitoring prototype and integration of the prototype in e-Health use case is pending.
DO-4.3.14	Access control	2	Different users and roles should have different level of access. Need support for role based and user-based access control. It would also be interesting to look at context aware authorization (e.g., in an emergency the access may be different than in normal operation)	H	M	CAAC	**(Partly covered)** The Context-Aware Access Control tool is an evolution of the authentication and authorization mechanisms provided by Evidian Web Access Manager (WAM) intended for the Internet of Things. It provides access control for both users and devices. The context-awareness aspect of these mechanisms is currently defined in a high-level specification. The next step is to implement this aspect.
DO-4.3.15	Authentication	2	Various kinds and levels of authentication need to be supported both at the edge side and cloud side. Support for two factor authentication (or similar level) is mandatory for a set of scenarios in the digital health domain	H	M	IoT app	Not to be addressed by WP4.
DO-4.3.16	Secure data transmission	2	Confidentiality, integrity, and authentication across IoT, edge and cloud is needed.	H	M	Secure protocols, S&P Monitoring Enabler	**(Partly Covered)** The IoT Platform used in e-Health is a security enabled gateway (TellU Gateway) and the protocol used is encrypted BTE. Yet, the analysis of monitored data to detect anomalies and notification functionality is

[6] CAAC – Context Aware Access Control.

							under development in the S&P Monitoring prototype.
DO-4.3.17	Communication need to be trustworthy in the sense of reliability, availability, integrity and privacy	2	The trustworthiness aspects of communication within digital health is significant for example, in order to not miss any notifications or alarms, you should be always connected to support emergency situations when they occur, the integrity of data is severe, and privacy need to be ensured as there is typically person sensitive data involved	H	M	Secure protocols, S&P Monitoring Enabler	**(Partly Covered)** The IoT Platform used in e-Health is a security enabled gateway (TellU Gateway) and the protocol used is encrypted BTE. Yet, the analysis of monitored data to detect anomalies and notification functionality is under development in the S&P Monitoring prototype.
DO-4.3.18	Monitoring, Diagnose information and failure detection	2	The system should continuously monitor system performance, suspicious behavior and failures. The monitored data should be analyzed to provide informative and understandable diagnosis	M	R	S&P Monitoring Enabler	**(Partly Covered)** The analysis of monitored data to detect anomalies is under development.
DO-4.3.19	Full end-to-end security	2	Support for security across the IoT, edge and cloud space from the medical device, through the gateway and all the way to the target stakeholders (e.g., hospitals, Electronic patient journal etc.) is needed	M	R	Secure protocols, S&P Monitoring Enabler, S&P Control Enabler, CAAC.	**(Partly Covered)** The IoT Platform used in e-Health is a security enabled gateway (TellU Gateway) and the protocol used is encrypted BTE. Yet, the analysis of monitored data to detect anomalies is under development in the S&P Monitoring prototype.

3 Design support to IoT system Security, Privacy and Resilience

This section describes the methods and mechanisms that will be offered by ENACT as security-by-design, privacy-by-design and resilience-by-design techniques to be adopted in IoT system development.

The ENACT support to security and privacy at design time is focused on: i) mechanisms for the specification of both the requirements of the IoT system components with respect to these two aspects, and ii) mechanisms for the specification of the necessary controls (external or internal to the IoT system) that ensure the requirements are met. The approaches proposed for privacy requirements and controls specification are the same as those of security requirements and controls specification, which eases the engineering of the IoT system since both aspects are addressed similarly in the DevOps process. Therefore, Section 3.1 provides the description of the mechanisms proposed for security-by-design and privacy-by-design together. Finally, Section 3.2 describes the diversity mechanisms that will be developed for diversity-by-design techniques.

3.1 IoT Security-by-design and Privacy-by-design mechanisms

In this section we describe the different mechanisms offered by ENACT solution to address trustworthiness in SIS design from the point of view of security and privacy-aware design. First, we explain the process supported by ENACT for the SIS security and privacy requirements expression (section 3.1.1). Second, we explain the procedure and mechanisms for identification and definition of the security and privacy controls in the SIS design (section 3.1.2), with a special focus (section 3.1.2.1) on the support provided in GeneSIS modelling language to this aim.

3.1.1 IoT Security and Privacy requirements specification

In ENACT the IoT system specification activity consists in the system modelling supported by GeneSIS language and tool (see D2.2 for more details). Therefore, ENACT proposes to address the system security and privacy requirements definition together with architectural and deployment requirements definition as part of the GeneSIS model of the system. To this aim, ENACT has profited from the recent advances on CloudML extensions [1] to describe security requirements at component level in distributed multi-cloud applications evolving them towards IoT systems. ENACT WP2 has enriched GeneSIS to include concepts for the specification of security and privacy requirements similar to those of CloudML oriented formalisms (for example Provided and Required Ports with Security Capabilities, see details in D2.2).

The GeneSIS language, similarly to CloudML, enables to capture system requirements at high-level of abstraction and independently of the actual providers and services used. GeneSIS as CloudML is Cloud provider independent and is service independent. The transformation from CloudML to GeneSIS models is fully possible, while the transformation from GeneSIS to CloudML models is possible but only for the Cloud parts of the system architecture, because some aspects of GeneSIS models (e.g. Controller, Devices, Hardware Capabilities, see details in D2.2) cannot be represented in CloudML.

Therefore, the description of security (availability, integrity and confidentiality) and privacy (personal data protection and security measures applied to personal data) aspects of IoT systems will be addressed in ENACT by specifying within the system GeneSIS models which are the security and privacy controls associated to the components, be they internal or external to the system. By internal components we refer to those developed or owned by the DevOps team developing the IoT system and which full control is in hands of the team, whereas external components would be those services used by the IoT system (invoked, used as deployment platform, etc.) but which control resides in a third-party service provider.

As described in deliverable D2.2, the identification of security and privacy controls required and provided by the components deployed in the IoT environment will require a previous risk analysis combined with the self-assessment of the internal components.

The **self-assessment** is a necessary step towards understanding which security and privacy measures are being implemented in the components under development. Different approaches of performing this analysis can be adopted, such as the one proposed by MUSA [2], where the component developers or product owners would go through a guided checklist indicating which security and privacy controls from standard control families are offered by the component or service under analysis. This assessment will therefore result in the list of controls already available in the component and thus not needed to be requested to any third-party. In MUSA, the NIST SP 800-53 rev 4 [3] security control family was used, but other control families can be adopted, according to the interest of the organisation, such as ISO/IEC 27002 [4], ISO/IEC 27017 [5], ISO/IEC 27018 [6], CSA CCM [7], etc.

Self-assessment is a best practice promoted in the community of security and privacy experts in different fields. At cloud layer of the IoT System, the STAR program by Cloud Security Alliance promotes the use of the Consensus Assessments Initiative Questionnaire (CAIQ) [8] that provides a set of Yes/No questionnaire for assessing the status of the controls in the Cloud Controls Matrix. In web application context, the Application Security Verification Standard (ASVS 4.0) [9] proposed by OWASP could be used as the basis of the security features check. Similarly, Berkley DB best Practices [10] may help in assessing database security features and correct implementation of countermeasures in very specific contexts.

The self-assessment is a per component process and once all internal components in the IoT system are assessed, the DevOps team will have a clearer picture of security and privacy posture of the components integrating the system in form of a well-structured collection of offered security controls and privacy controls, which will be an input to the Risk Assessment process in order to identify whether there is any missing control for the system security and privacy behaviour assurance.

It is interesting to note that from all existing standard security control families, only NIST control family in its latest version rev 5 Draft [11] and the ISO/IEC 27552 [12] (under development) include actual privacy controls related to the protection of personal data. Furthermore, in addition to distinguishing between privacy and security controls, NIST SP 800-53 rev 5 Draft [11] identifies also *joint* controls as those able to address system security and privacy requirements at a time. At this point, one should bear in mind that privacy, even according to the GDPR, is intrinsically related to security mechanisms applied in personal data protection by services or systems. Article 5.1(f) of GDPR, requires that personal identifiable information (PII) *shall be processed in a manner that ensures appropriate **security** of the personal data, including protection against unauthorised or unlawful processing and against accidental loss, destruction or damage, using appropriate technical or organisational measures ('integrity and confidentiality').*

For external third-party components or services, it is necessary to learn in advance which are the offered security and privacy controls or policies, which usually depend on the type of service and/or supporting device (e.g. the available encryption modes in a gateway, the authentication mechanism used by a sensor, if any, etc.). This information is generally provided by the service/component (or device) provider, even if it not usually described in form of standard security or privacy controls. It is the task of the DevOps team to try to homogenise the way the offered security and privacy mechanisms are specified for external components with that of the internal components, so the overall security and privacy posture of the IoT system is understood and can be controlled easily.

The **security and privacy risk analysis** in ENACT is supported by the Risk Management tool (see full description in deliverable D2.2) which will allow the DevOps team to reason over system assets requiring protection, learn which are their vulnerabilities and potential cyber incidents and attacks against these assets. At IoT system level, the assets would refer to system components. Therefore, the risk assessment will identify the most severe cyber risks over the components according to the IoT

system architecture, communications between the components and types of data exchanged. The result of this process will be the identification of the relevant cyber threats that need to be addressed and the possible mitigation means. As part of these mitigations, the required organisational procedures (e.g. the inclusion of privacy aspects in the data quality management procedures, the implementation of Privacy Impact Assessments, the security and privacy trainings, etc.) and required system mechanisms may be identified (such as the adoption of access control, the inclusion of a vulnerability scanning, the monitoring of a specific interfaces, the de-identification of personal data, etc.).

3.1.2 IoT Security and Privacy controls specification

As explained above, different control families can be used to express the security and privacy controls of the systems in a standard way. These include but are not limited to NIST SP 800-53 rev 5 Draft [11], ISO/IEC 27002 [4], ISO/IEC 27017 [5], ISO/IEC 27018 [6], CSA CCM [7], etc. These catalogues aid the formal specification of required and/or offered (provided) capabilities of the IoT System as a whole and/or its individual components and the organisation providing it/them. In those cases where personally identifiable data is processed by the IoT system or any of its components, privacy controls need to be adopted.

As the major exponent of fine-grained controls catalogue publicly available for free and internationally adopted, the NIST SP 800-53 rev 5 Draft [11] collects 1026 controls identified as *security* controls, privacy controls and *joint* controls (which serve either security or privacy or both). A total of 161 privacy related controls are identified by NIST SP 800-53 rev 5 Draft [11] from 12 different areas or groups (named *control families* by NIST), namely: AC - Access Control, AT - Awareness and Training, AU - Audit, CA - Continuous Assessment, CM - Configuration Management, CP - Contingency Planning, IA - Identification and Authentication, IP - Individual Participation, IR - Incident Response, MP - Media Sanitization, PA - Privacy Authorization, PL - Planning, PM - Project Management, RA - Risk Assessment, SA - System and Services Acquisition, SC - System and Communications, SI - System and Information Integrity. From these, 60 controls are *privacy* controls and 101 are *joint* controls.

NIST also distinguishes between organisational controls (“O” – a control implemented by a human in the organization through nontechnical means) or system controls (“S” – a control typically implemented by an organizational system through technical means) or controls implemented by either or the combination of both nontechnical and technical means (“O/S”). The privacy related “S” and “O/S” controls identified by NIST rev 5 Draft are only 12 and 14 respectively, summing up a total of 26 technically implementable privacy controls. Therefore, according to NIST, most of the means for tackling with privacy assurance in the systems, reside at the organizational or procedural level, not at system level. In the following table, we summarize the system level controls related to privacy, trying to stick as much as possible to the NIST guidance on the controls. In the table, column 3 identifies privacy controls marked as “P” and joint controls marked as “J”, while system controls are marked as “S” and controls that can be addressed with technical and/or non-technical means are marked as “O/S”.

Table 3. System level controls related to privacy

Control ID	Control in the NIST SP 800-53 Revision 5 Draft	P vs. J[7]	S vs. O/S[8]	Summary
AC-16(1)	Security and Privacy Attributes | DYNAMIC ATTRIBUTE ASSOCIATION	J	S	Both security and privacy related properties or characteristics (attributes) of data subjects and data objects should be considered.
AC-16(2)	Security and Privacy Attributes | ATTRIBUTE VALUE CHANGES BY AUTHORIZED INDIVIDUALS	J	S	
AC-16(3)	Security and Privacy Attributes | MAINTENANCE OF ATTRIBUTE ASSOCIATIONS BY SYSTEM	J	S	
AC-16(4)	Security and Privacy Attributes | ASSOCIATION OF ATTRIBUTES BY AUTHORIZED INDIVIDUALS	J	S	

[7] P=Privacy control and J=Joint control

[8] S=System control and O/S= Organization and/or System control

AC-16(5)	Security and Privacy Attributes | ATTRIBUTE DISPLAYS FOR OUTPUT DEVICES	J	S	
AC-16(8)	Security and Privacy Attributes | ASSOCIATION TECHNIQUES AND TECHNOLOGIES	J	S	
AC-16(11)	Security and Privacy Attributes | AUDIT CHANGES	J	S	
AU-12(4)	Audit Generation | QUERY PARAMETER AUDITS OF PERSONALLY IDENTIFIABLE INFORMATION	P	S	Audit of query parameters within systems for datasets that contain personally identifiable information.
PA-3(2)	Purpose Specification | AUTOMATION	P	S	Automated mechanisms may be used to support records management of authorizing policies and procedures for personally identifiable information.
SC-7(24)	Boundary Protection | PERSONALLY IDENTIFIABLE INFORMATION	P	O/S	Ensure that personally identifiable information is used or transmitted only in accordance with established privacy requirements.
SC-16	**Transmission of Security and Privacy Attributes**	J	S	Associate organization-defined security and privacy attributes with information exchanged between systems and between system components.
SI-4(25)	System Monitoring | PERSONALLY IDENTIFIABLE INFORMATION MONITORING	P	O/S	Automate the monitoring of (a) unauthorized access or usage of personally identifiable information; and (b) the collection, creation, accuracy, relevance, timeliness, impact, and completeness of personally identifiable information.
SI-6	**Security and Privacy Function Verification**	J	S	Verify the correct operation of not only security but also privacy functions of the system and notify and react in case anomalies are detected.
SI-15(1)	Information Output Filtering | LIMIT PERSONALLY IDENTIFIABLE INFORMATION DISSEMINATION	P	O/S	Limit the dissemination of personally identifiable information to organization-defined elements identified in the privacy risk assessment and consistent with authorized purposes.
SI-18	**Information Disposal**	P	O/S	Disposal or destruction of information (originals, copies, archived records, including system logs) that may contain personally identifiable information.
SI-19	**Data Quality Operations**	P	O/S	Perform operations on personal identifiable information to ensure its quality in terms of accuracy, relevance, timeliness, impact, completeness, and de-identification.
SI-19(1)	Data Quality Operations | UPDATING AND CORRECTING PERSONALLY IDENTIFIABLE INFORMATION	P	O/S	
SI-19(2)	Data Quality Operations | DATA TAGS	P	O/S	
SI-19(3)	Data Quality Operations | PERSONALLY IDENTIFIABLE INFORMATION COLLECTION	P	O/S	
SI-20	**De-Identification**	P	O/S	Personally identifiable information shall be removed from datasets when it is not (or no longer) necessary to satisfy the
SI-20(1)	De-Identification | COLLECTION	P	O/S	
SI-20(2)	De-Identification | ARCHIVING	P	O/S	
SI-20(3)	De-Identification | RELEASE	P	O/S	

SI-20(4)	De-Identification | REMOVAL, MASKING, ENCRYPTION, HASHING, OR REPLACEMENT OF DIRECT IDENTIFIERS	P	S	requirements envisioned for the data.
SI-20(5)	De-Identification | STATISTICAL DISCLOSURE CONTROL	P	O/S	
SI-20(6)	De-Identification | DIFFERENTIAL PRIVACY	P	O/S	
SI-20(8)	De-Identification | MOTIVATED INTRUDER	P	O/S	

The control IDs that include a number between brackets refer to control enhancements which complement basic controls. Therefore, most of the privacy controls that can be addressed at system level are enhancements of other basic controls. Following this analysis, a total of 11 controls are addressed by technical measures (at system level) and of those, only 5 controls are basic controls and not enhancements: SC-16 Transmission of Security and Privacy Attributes, SI-6 Security and Privacy Function Verification, SI-18 Information Disposal, SI-19 Data Quality Operations, SI-20 De-Identification.

3.1.2.1 IoT Security and Privacy controls specification in GeneSIS

In this subsection, we present the language and tool support in GeneSIS for specifying IoT security and privacy controls in the deployment models of smart IoT systems (SIS) and deploying them together with the components of SIS. We show below some representative examples of the main features that GeneSIS supports for deploying security and privacy mechanisms. The features described in this subsection are derived from the collaboration between WP2 and WP4. More details of the GeneSIS tool and different features for supporting the deployment of IoT applications together with security and privacy mechanisms can be found in D2.2.

A main requirement for GeneSIS deployment model regarding support for security and privacy specification is that any software component can require to be deployed together with a specific security/privacy component that provides a certain security and privacy capability. GeneSIS can choose the security and privacy controls from the catalogues of security and privacy controls mentioned in Section 3.1.2 above. We basically organise the security and privacy controls into two main catalogues as shown in Figure 2. The first catalogue contains the ENACT-driven security and privacy controls that are tailored for the advanced supports regarding trustworthiness. The other catalogue consists of other security and privacy controls in the literature that can also be used by the GeneSIS. Figure 2 shows how the two catalogues used as input of GeneSIS. Driven by the Risk Management process, a DevOps team working on the deployment model using GeneSIS can select the security and privacy controls with required capabilities from the catalogues for risk treatment. GeneSIS allows the chosen security and privacy controls to be specified for being deployed together with the other components of SIS.

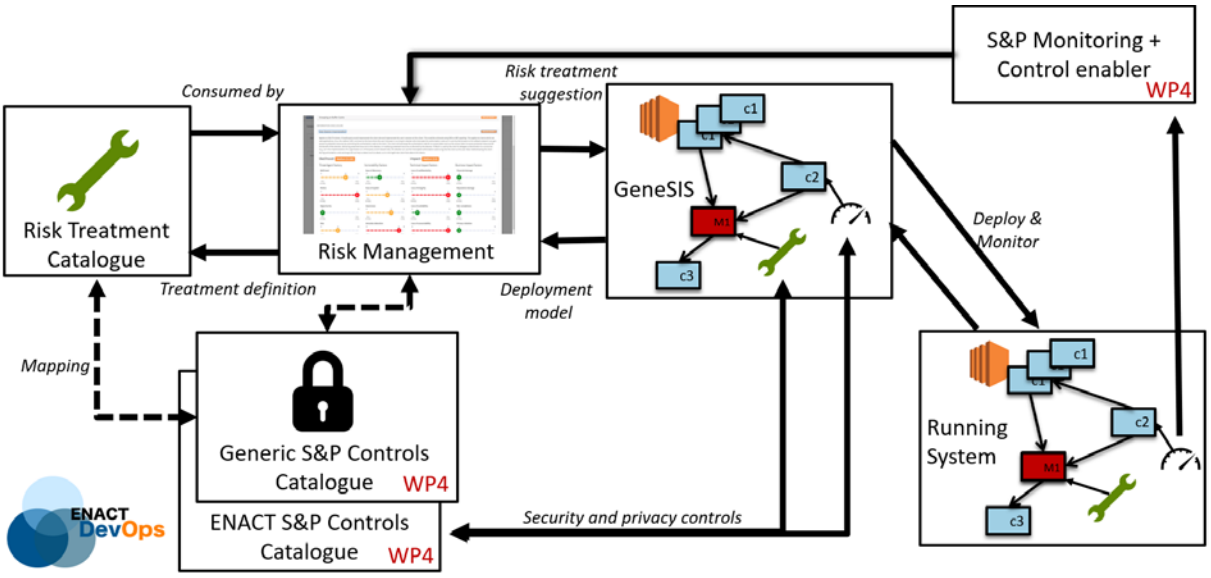

Figure 2. The security and privacy controls of WP4 as input for GeneSIS

In D2.2, we have described the current support of GeneSIS for deploying security and privacy controls together with the IoT software components of a SIS. For example, an IoT Energy Efficiency application that reads sensor data must only do so according to an access control policy, or context-based access control mechanisms. It is important that the IoT applications can only receive sensors data that they are allowed to access and can only send commands to the actuators that they are allowed to control, in a dynamic context. For example, for security (and privacy) reasons, the IoT Energy Efficiency application can receive sensors data about temperature and light, but not live video from cameras. The IoT Energy Efficiency application can control heating system or window blinds, but NOT in case of fire alarm. For privacy reasons, camera data cannot be sent out of the gateway that directly connects to the camera. To address such security and privacy requirements for the IoT Energy Efficiency application, a DevOps team working on the IoT Energy Efficiency application can select and adapt the security and privacy controls with capacities satisfying the requirements, configure them with required policies and use GeneSIS to specify them to be deployed together with the components of the IoT Energy Efficiency application.

The examples of context-based access control and privacy policy enforcements discussed above are implemented at application level, normally engineered for a specific smart IoT system. The enforcement of security and privacy policies will be in place after GeneSIS deploys the required security and privacy mechanisms together with IoT applications. Therefore, the main requirement for GeneSIS is the following. When deploying the IoT applications, we also need to deploy security and privacy mechanisms together with policies that must be enforced for those IoT applications. From the deployment point of view, GeneSIS should support specifying 1) the capabilities (security/privacy) of each component, and 2) the ports associated with components. The port specification is useful for security mechanisms such as a Security Gateway that only exposes certain ports for controlled accesses to protected sensors or actuators. Figure 3 shows the concepts of ports and capabilities of components that are supported by GeneSIS. With these supports, each IoT component that requires a certain security or privacy control can only be 'linked' to components providing the necessary capabilities using a *Communication* link, via the provided ports. When GeneSIS deploys a security or privacy component/control, it means configuring it, including with security or privacy policies if any.

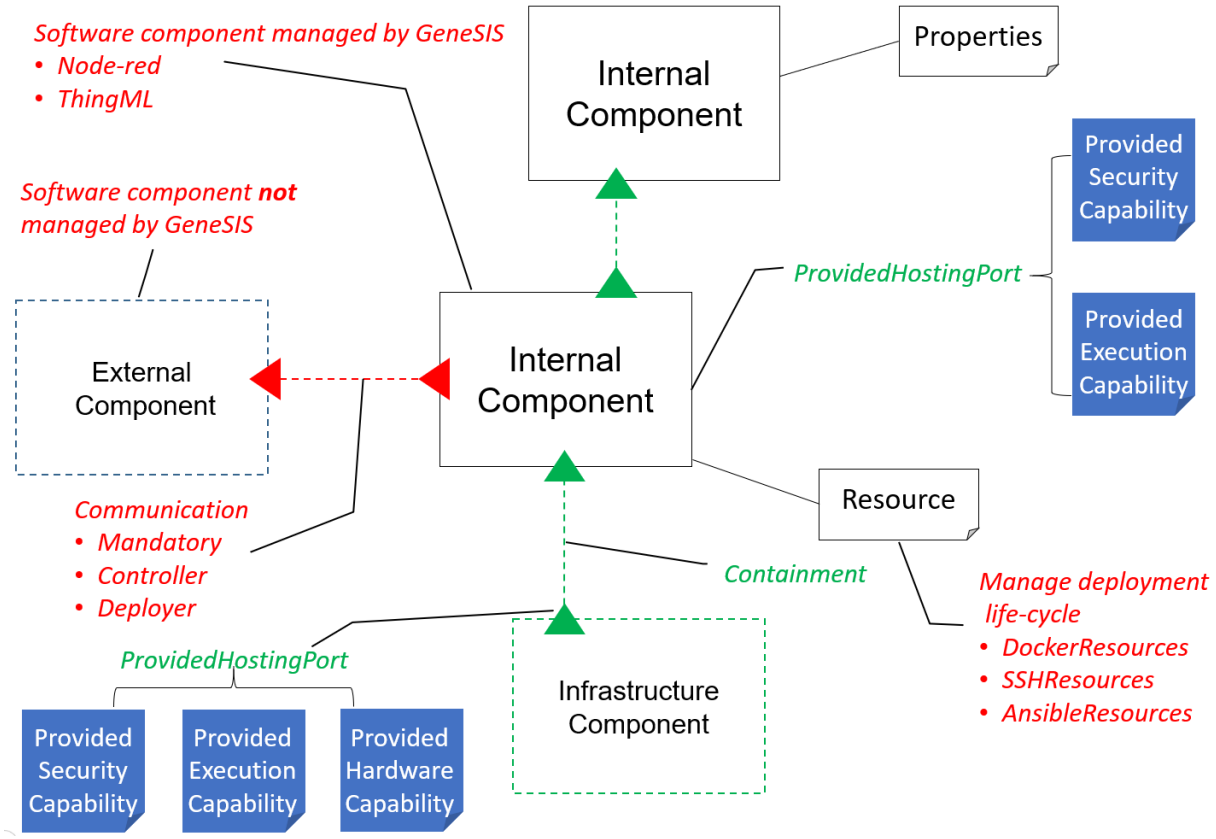

Figure 3. The generic concepts of components in GeneSIS with capability specification support

The main requirement presented above is the core feature for GeneSIS to support the deployment of security and privacy mechanisms. More "advanced" requirements are also considered by GeneSIS. For example, location constraints such as how far is the host of a security module from the sensors or actuators might be expressed. Such constraint reflects the fact that a security module may need to be a local node to a gateway to guarantee the performance of the authentication process. Moreover, a software component may have a constraint to be deployed only two “steps” away from the data source or actuator. For instance, this may be due to a privacy policy specifying that this specific software component that processes sensitive data cannot be deployed more than two network levels away from the data source.

3.2 IoT Diversity-by-design mechanisms[9]

The ENACT IoT Diversity-by-design tool consumes as inputs and outputs some of the design-time artefacts in the ENACT project, i.e., the deployment model specified in the GeneSIS modelling language and the behaviour model specified in ThingML. In particular, the tools take as input a single deployment or behaviour specification and generate multiple diverse specifications. Within a DevOps context, it is important and necessary to keep the diversity generation fully automatic, instead of relying on developer's manual effort to main diverse systems (such as the traditional N-Version Programming approach[10]). Developers can focus on a single line of code to achieve frequent iteration, and the diversification tool, as part of automatic building step, will generate diversified version automatically.

In practice, the mechanism is implemented as independent tools: the architecture diversifier and the code diversifier, handling the deployment model and the behaviour specification, respectively. At the current stage, the architecture diversification is focused on diversifying the orchestration of reusable blocks, and the code (behaviour) diversifier is focused on the diversification of communication protocols. In the first period, we focus on the **code diversification** at the protocol level.
Automated diversity is a promising means of mitigating the consequences of a security breach. However, current automated diversity techniques operate on individual processes, leveraging mechanisms available at the lower levels of the software stack (in operating systems and compilers), yielding a limited amount of diversity. In this section, we present a novel approach for the automated synthesis of diversified protocols between processes. This approach builds on a) abstraction, where the original protocol is modelled by a set of communicating state machines, b) automated synthesis, applying mutation operators onto those protocols, which produces semantically-equivalent, yet phenotypically-different protocols, and c) automated implementation of these protocols through code generation.

The tool is currently in an experimental stage. Automatic diversity of communication protocols is a novel technology, yet without convincing implementation and applications, to the best of our knowledge. Therefore, our focus is currently on the theoretical feasibility of the idea and the experimental evaluation of its effects. In the next step, we will improve the user experience of the tool and its applicability to practical scenarios.

Mass-produced software applications denote clonal applications, with thousands or millions of identical siblings. Think of, for example, a popular mobile application installed on millions of mobile phones, or software embedded into a widely-used connected device. To mitigate the risks of such large mono-cultures, diversity is typically automatically introduced either in a generic way, typically at the OS level, oblivious of the actual logic and semantics of the software, or in some very specific places, typically low-level libraries reused across applications, in order to improve security. This leaves most of the actual business logic unchanged, unaffected by the diversity. In addition, diversity often affects individual processes, but leaves the communication between processes intact.

[9] Source code of this tool can be found here: https://github.com/SINTEF-9012/thingml-diversifier
[10] Avizienis, Algirdas. "The N-version approach to fault-tolerant software." IEEE Transactions on software engineering 12 (1985): 1491-1501.

A more holistic approach to diversity is challenging. Consider a typical client-server application, where multiple clients interact with a server, and where each client has a different implementation, and a different way of communicating with the server. This would significantly hinder a hacker, be it a human being or a machine, when attempting to generalize an attack through all possible protocols. This would make large-scale exploits a time-consuming and costly endeavour for hackers. Yet, the engineering, e.g., the production, maintenance and integration, of such levels of diversity raises several challenges. How to ensure that each implementation still behaves as specified? How to ensure that each client is still able to communicate with the server, without information loss or distortion? How to ensure that different clients are fundamentally (i.e., sufficiently) different, and not merely cosmetically different? How to keep the development and operation costs of a diversified system significantly lower than the cost of mitigating large scale attacks?

We have seen that abstraction, synthesis and automated implementation can yield a convincing solution to introduce a wide diversity into protocols, for example between a device and a gateway, or a web/mobile app and a server. This approach:

- abstracts protocols into a) a structural view describing the messages to be exchanged, and b) a behavioural view based on state machines describing how those messages are exchanged between the participants, including sequencing and timing.
- combines and applies a number of atomic mutations to this protocol model, yielding a large number of diversified protocols, which operate differently, still with the same semantics.
- automatically implement protocols, diversified or not, by generating fully operational code targeting C, Go, Java and JavaScript, able to run on a wide range of platforms.

Our empirical assessment indicated that this approach implies a reasonable overhead in terms of execution time, memory consumption and bandwidth, fully compatible with the requirements of "mass-produced" software. We also showed that this approach could generate a significant amount of diversity. Our assumption was that this diversity would contribute to the diversity-stability hypothesis, i.e., this would make the whole ecosystem more robust by making it less likely for an exploit to propagate to the whole population. In other words, if the protocol between a specific client and the server could be observed, analysed and eventually understood, this would not systematically imply that all other diversified protocols could be understood following the very same procedure.

In this section, we briefly describe the mechanisms and the corresponding tools we developed to automatically generate the diverse protocols. Technical details and the experiment results can be found in our conference paper[11].

Our approach relies on ThingML for the specification of protocols. ThingML provides a way to formalize the messages involved in protocols, in a comparable way to what Protocol Buffer proposes. In addition, ThingML provides a mean to formalize the behaviour of protocols through state-machines. ThingML specifications are both human-readable and machine-readable, which makes it possible to analyse protocols at a high-level of abstraction and to fully automate the implementation of those protocols through code generation. In the next-sub-section, we present relevant aspects of ThingML on our motivating example.

We model communication protocols as a set of communicating state-machines, encapsulated into components. A protocol typically involves two roles: 1) a client, i.e., a device, a web-browser or a mobile app, and 2) a server i.e., a gateway or a cloud back-end. The clients and the server need to agree on a common API. Since communication is typically asynchronous in a distributed system, the common API is specified as a set of messages. Next, this API is imported by the client component and the server component, and the messages are organized into ports.

[11] Morin, Brice, Jakob Høgenes, Hui Song, Nicolas Harrand, and Benoit Baudry. "Engineering Software Diversity: a Model-Based Approach to Systematically Diversify Communications." In Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, pp. 155-165. ACM, 2018.

The ultimate goal of our approach is to diversify the wire image of protocols[12]. Diversifying the wire image of protocols basically means shuffling the sequence of bytes exchanged over the network e.g., turning the payloads while ensuring the interoperability between the client and the server. This comes with a number of constraints:

- The first byte of each block must be an identifier for the message encoded by this block. Though this is not an absolute requirement, it is a commonly-accepted practice as it allows to implement, or generate, efficient parsers with no need for extra memory and back-tracking.
- The order of messages should be preserved.

The model-based diversification process itself is depicted in Figure 4. The diversifier is a 1-to-n, endogenous transformation. It takes a ThingML protocol model as input and generates n ThingML protocol models as output. A key benefit of this design choice is that the whole ThingML tool-chain can be reused as-is on those diversified protocols. The ultimate output of this process is a set of binaries, for any of the languages supported by ThingML, implementing the diversified protocols. Optionally, these binaries can be automatically packaged as Docker components for facilitating their deployment.

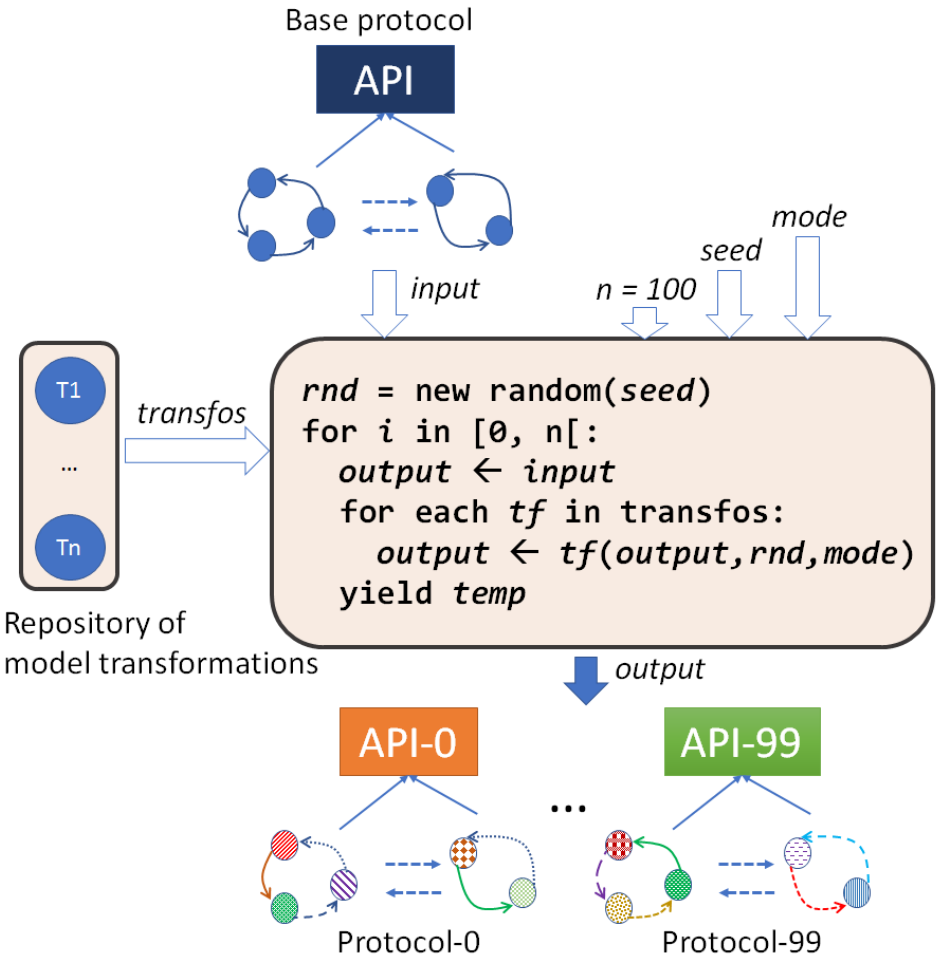

Figure 4. Model-based diversification process

The diversification process is configured according to the following parameters:

1. Seed: the seed for the random number generator used internally by the diversifier. Given one seed and one input protocol, the diversifier produces repeatable outputs.

[12] https://tools.ietf.org/html/rfc8546

2. Transformations: the sequence of model transformations to apply on the input model. Each transformation takes a valid ThingML model and produces more ThingML models, all of which are valid but different from the original one. The technical details of these transformations can be found in a published paper [13].
3. Mode: Most transformations introduce a random choice. Such a choice can be:
    - Static: in this mode, the diversifier generates different implementations for the input protocol. However, all the diversity is fixed by the diversifier itself - i.e., the protocol will not evolve over time.
    - Dynamic: in this mode, the diversifier still generates different implementations. In addition, some diversity still remains open at runtime, so as to allow the protocol to change over time (within some boundaries) and act as a moving target defence.
4. The number of diversified protocols to be generated.

The automatic generation of both client and server-side code is an essential feature of our MDE process for the diversification of communication. Here, we rely on the compilers available as part of the ThingML Framework. The diversified protocol models being “plain old ThingML specifications”, the existing compilers. This way we can generate code in C (for resource-constrained microcontrollers or Linux), Java, JavaScript (for browsers or Node.JS) or Go for both the client and the server sides.

4 Operation support to IoT system Security, Privacy and Resilience

This section describes the support offered by WP4 initial Enabler prototypes to IoT system operation with respect to ensuring a secure, resilient and privacy-respectful behaviour.

While monitoring support will be offered by the so-called Security and Privacy Monitoring and Control Enabler, resilience support will be provided by the Robustness and resilience Enabler, also named Code Diversifier or DivEnact. Both enablers have been designed with a special focus on the large scale IoT systems, i.e., systems that comprise large number of sub systems, each of which may serve a different customer.

4.1 Security and Privacy Monitoring mechanisms

Addressing security and privacy assurance in IoT systems is a challenging task due to the complexity of multi-layer nature (thing, edge, Cloud, network, etc.) of IoT systems and heterogeneity of system components from system to system. In ENACT WP4 the research has focused on how to effectively offer to DevOps teams the means to be able to ensure the continuum of security and privacy properties of the system, while keeping the mechanisms independent from system particular implementation as much as possible.

To this aim, ENACT WP4 has designed and developed the Security and Privacy Monitoring and Control Enabler as support to the tasks of controlling (personal) data protection and secure communications all along the operational life of the SIS. The first step in the control is the continuous monitoring of the status of the system security and privacy. The Monitoring part of the Enabler is described in this section 4.1 while the Control part is described in next section 4.2.

As it will be seen, for the Monitoring service in ENACT we have leveraged a number of state-of-the-art technologies which we have integrated together to offer a complete service which main innovation resides in three major axes:

- the **richness of the anomaly detection** by correlation of **multi-layer data** from the SIS. Through exploiting the data captured by multiple probes or monitoring agents deployed in network, application and system layers of the SIS, the anomaly detection under development will offer a holistic view of situational awareness in the SIS and enable the detection of sophisticated attacks or incidents, allowing for easier identification of root-cause.
- the **flexibility and extensibility of the architecture** to address the needs of different types and sizes of SIS. In cases where it is not possible to deploy all types of monitoring agents, because of unreachability to a layer (network, system or application) the solution design enables to use only the types adoptable and still detect anomalies with the information available. The design allows also to focus on specific types of anomaly events in case it is desired.
- the **scalability** of the solution. By relying on Big Data technologies, we have guaranteed that the Monitoring tool is fully elastic to be able to scale in large-scale IoT system scenarios where hundreds or thousands of monitoring agents need to be deployed.

4.1.1 Purpose

According to multiple reference organisms such as NIST or ENISA, an imperative activity applied to an IoT system should be the Security Monitoring activity [14]. It includes protocols and data formats that enable the ongoing, automated collection, monitoring, verification, and maintenance of software, system, and network security configurations, and provide greater awareness of vulnerabilities and threats and the overall status of the security of the system.

The Security and Privacy Monitoring tool in ENACT allows the IoT system operator to monitor the security and privacy status of the IoT system at different layers. The tool will capture and analyse data from multiple and heterogeneous sources such as raw data from network, system and application layers, as well as events from security monitoring components such as IDS/IPS. All the data will be processed and displayed in a common dashboard, which includes processed information in form of alerts, statistics and graphs.

4.1.2 Architecture

Figure 5 shows the architecture details of the ENACT Security & Privacy Monitoring and Control Enabler. Specifically, the monitoring related components are represented in blue and purple, while the control related components are represented in green. The latter ones are described in more details in section 4.2.2.

The ENACT Security & Privacy Monitoring Enabler, in short, Monitoring Enabler, captures raw data from the SIS through multiple distributed probes, named *agents*. The distributed monitoring mechanism relies on three types of monitoring agents that can retrieve raw data from the SIS: (i) **network agent**, which captures network traffic data; (ii) **system agent**, which can capture data related to activities of processes on the devices of the SIS; and (iii) **app agent**, which can capture operational data of the SIS. The app agent in ENACT has been implemented as a special mechanism to work with SMOOL IoT Platform that will be used in the ENACT Smart building use case. This mechanism will be continuously listening to all communications in and out through the SMOOL IoT Platform. See section 4.2.2 for more information on SMOOL and the implemented app agent.

The Monitoring Enabler inputs are further complemented by an open source **Network Intrusion Detection System (NIDS)** which will also capture network data and will generate security events based on pre-defined rules. These pre-defined rules are settled by connecting to open source and/or commercial databases with knowledge of the well-known cyber security attacks. Moreover, the rules can be enhanced by the Monitoring Enabler user at any time; the only restriction is to follow the common format of defining the rules [16].

Additionally to the mentioned inputs, the Monitoring Enabler can also acquire events from other Security & Privacy Control Enabler agents deployed in the SIS. For instance, the Policy Decision Point (PDP) control agent described in section 4.2.4 can send its events to the Monitoring Enabler.
All the raw IoT data captured by the Monitoring Enabler agents as well as the events generated by the IDS or the Security & Privacy Control Enabler agents are sent to a **streaming bus** that has a two-fold objective. First, the bus allows to acquire all the data in streaming and avoid a bottleneck at next phases of pre-processing of the IoT data. And second, it allows to send back the pre-processed IoT data and handle publish-subscribe threads related to the data handled by the streaming bus (pre-processed IoT data and events generated by different agents and components of the Security and Privacy Monitoring and Control Enabler).

After the acquisition of the raw IoT data and generated events, the next phase is to parse and do the required pre-processing of the acquired data. This is handled by the **Data Capturing and Parser** component which generates the pre-processed IoT data and sends it both to the **IoT data storage** infrastructure and back to the Streaming bus. The latter feedback to the Streaming bus aims to provide with streaming near real-time data to the **Anomaly detection** component at prediction phase.

Furthermore, the Anomaly detection component can gather pre-processed IoT data from the IoT data storage at training phase. The Anomaly detection component will be trained using different Machine Learning algorithms and different models will be created for the prediction phase. The objective is to complement the existing IDS with other anomaly detection mechanisms for cyber security threats.

Finally, the **Monitoring dashboard** will display all acquired data and generated events in a user-friendly manner in form of alerts, statistics and graphs.

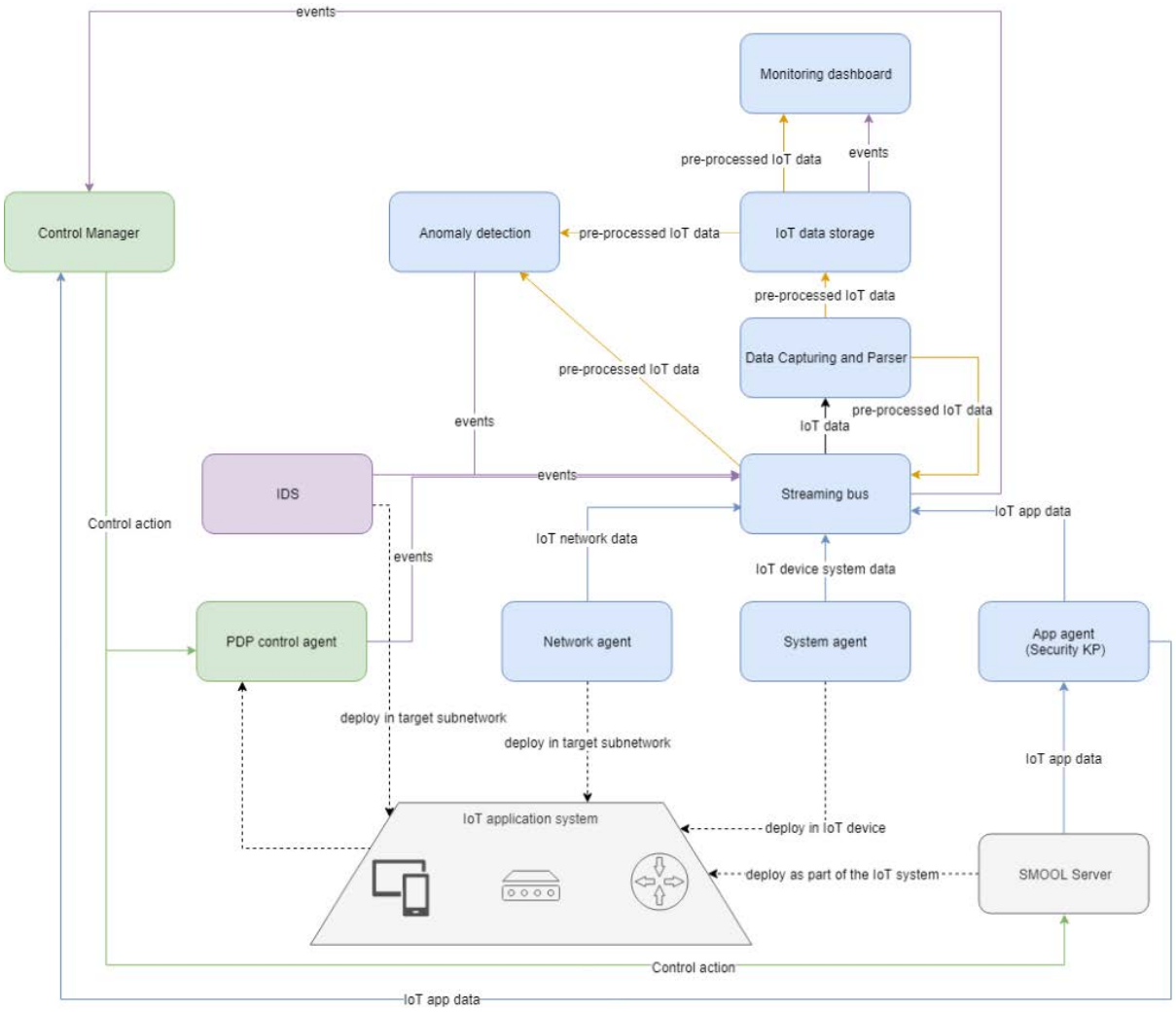

Figure 5. Architecture of ENACT Security & Privacy Monitoring and Control Enabler

4.1.3 Detailed design

The following sections explain more details of the Monitoring Enabler components. As introductory information, the Monitoring Enabler has been designed to be able to perform highly scalable monitoring and is based on Elastic stack provided technologies (i.e. Beats, Logstash, Elasticsearch and Kibana) [17].

4.1.3.1 Monitoring agents

The monitoring agents of the Monitoring Enabler are probes distributed in different components and layers of the SIS which are aimed at capturing raw data from the SIS that can be used in next phases to identify security events and detect security anomalies.
There are three types of monitoring agents that can be deployed in a SIS:

1. *Network agent*: Its objective is to acquire raw network traffic data of the SIS. It is based on two main sub-components: (i) t-shark [18], the core network protocol analyser of Wireshark, the most used network traffic capturer and analyser; (ii) and Filebeat [19] , which offers a lightweight way to forward and centralize the files with the acquired network data. It is crucial at this step to correctly define the mapping between the network attributes identified by the t-shark and the indexes that are required to be created in the IoT data storage (see section 4.1.3.5). Even if this mapping can be done dynamically, there is a limit of maximum dynamically created

indexes in Elasticsearch. And due to the huge amount of network attributes identified by the t-shark (since there are multiple protocols and network layers to capture and parse), defining properly the mapping is key. This is currently one of the main research topics of the monitoring task in ENACT.

2. *System agent*: Its objective is to acquire raw data regarding activities of users and processes of specific devices part of the SIS. It is based on Auditbeat [20] technology.
3. *App agent*: Its objective is to acquire operational data of the SIS. It is based on a SMOOL KP, a customized KP that is subscribed to all communications of the SIS handled by SMOOL platform. Since the SMOOL KP is able to understand the semantics of the operational data used by the SIS, the app agent can acquire this data already parsed to the specific semantics of the SIS. It also uses Filebeat, which offers a lightweight way to forward and centralize the files containing the IoT operational data.

4.1.3.2 Network Intrusion Detection System (NIDS)

There are several well-known open source IDS tools available in the market as well as several sources of ruleset of known cyber security attacks. The Monitoring Enabler aims at enriching the security event monitoring by including one of these tools to the solution and tailoring it to the networks in the ENACT use cases.

The Monitoring Enabler includes an open source signature-based Intrusion Detection System that is able to detect well-known security attacks. It is based on Suricata, which is capable of real time intrusion detection (IDS), inline intrusion prevention (IPS), network security monitoring (NSM) and offline pcap processing [21].

Suricata identifies events based on pre-defined rules that are settled by connecting to open source and/or commercial database sources with knowledge of the famous cyber security attacks [22].

4.1.3.3 Streaming bus

As mentioned in section 4.1.2, the streaming bus has a two-fold purpose. First, the bus allows to capture all the data sent by the monitoring agents and the IDS in streaming and avoid a bottleneck at next phases of pre-processing of the IoT data. Second, it allows to manage all the data required by the components of the Monitoring Enabler and handle the publish-subscription among the Monitoring Enabler components with regards to the pre-processed IoT data and events generated by different agents and components of the Security and Privacy Monitoring and Control Enabler.
It is based on Apache Kafka, an open source distributed streaming platform [23].

4.1.3.4 Data Capturing and Parser

The Data Capturing and Parser component allows to ingest data from heterogeneous sources simultaneously, and parses and transforms it to facilitate the incident and anomaly detection. It is based on Logstash, which is an open source, server-side data processing pipeline that ingests, transforms and sends the data to other component(s) [24]. In the case of the ENACT Monitoring Enabler, it ingests the data from the streaming bus and after transforming it, it sends it to two other components: the streaming bus and the IoT data storage.

4.1.3.5 IoT data storage

The IoT data storage stores all the acquired and pre-processed data in a NoSQL database. It is based on Elasticsearch, a distributed, RESTful search and analytics engine [25].
The IoT data storage defines multiple indexes based on the multiple sources of data (i.e. raw data from different monitoring agents and generated events by the IDS and/or control agents).
Following Figure 6 and Figure 7 show different indexes created by the acquired data, parsed and stored in the IoT data storage.

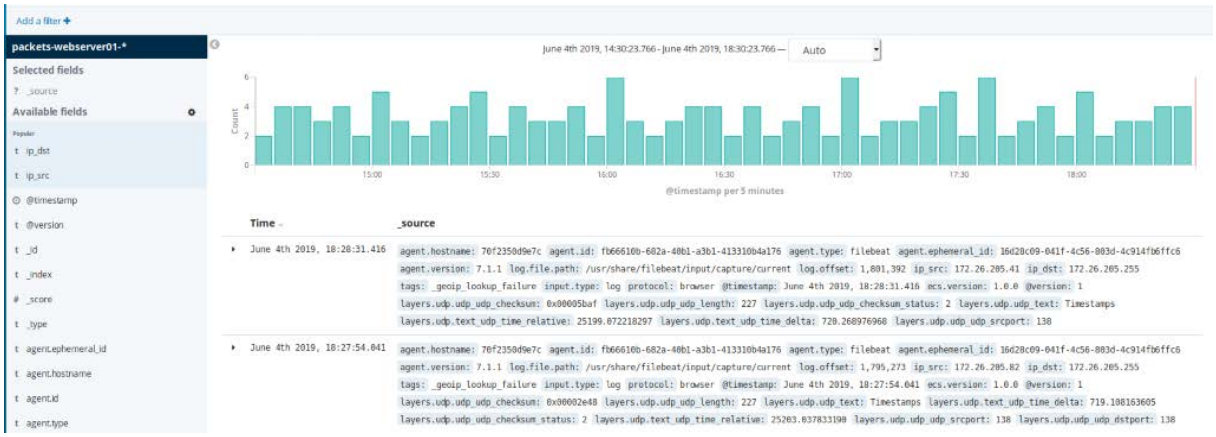

Figure 6. Example of indexes created in Elasticsearch

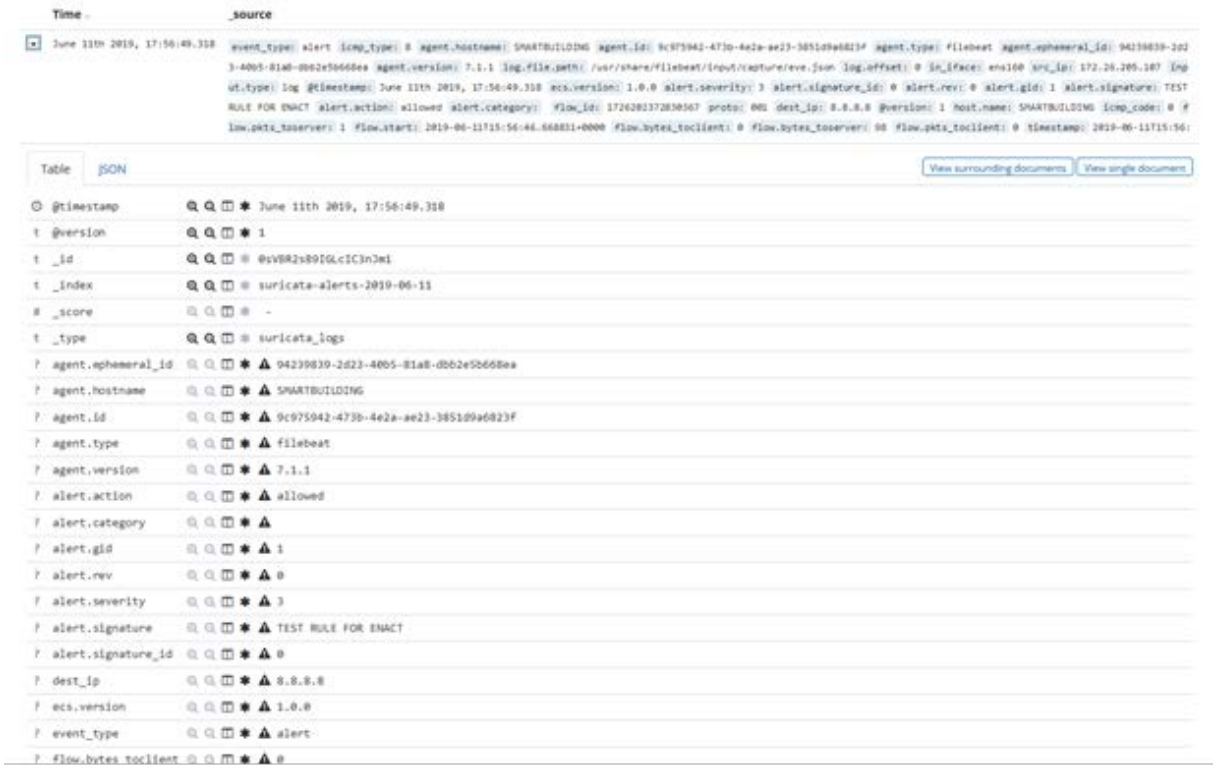

Figure 7. Example of indexes created in Elasticsearch

Next it is shown an entry in the database of a NIDS event, specifically an alert of “Scan Network”.

```
{
 "_index": "suricata-logs-2019-06-19",
 "_type": "suricata_logs",
 "_id": "XN15bmsB9IGLcIC3UR1T",
 "_version": 1,
 "_score": null,
 "_source": {
  "agent": {
   "hostname": "855db674acac",
```

```
  "id": "7c7a1b3a-c33b-4ee1-9bce-44248178af0a",
  "type": "filebeat",
  "ephemeral_id": "a61ad4e4-8255-4cc3-9efd-c46cba095d08",
  "version": "7.1.1"
 },
 "log": {
  "file": {
   "path": "/usr/share/filebeat/input/capture/eve.json"
  },
  "offset": 8621941
 },
 "tx_id": 0,
 "app_proto": "http",
 "src_ip": "172.26.252.100",
 "src_port": 60367,
 "in_iface": "enp2s0",
 "input": {
  "type": "log"
 },
 "@timestamp": "2019-06-19T06:42:51.860Z",
 "event_type": "alert",
 "ecs": {
  "version": "1.0.0"
 },
 "alert": {
  "severity": 3,
  "metadata": {
   "updated_at": [
    "2013_10_18"
   ],
   "created_at": [
    "2013_10_18"
   ]
  },
  "signature_id": 2017616,
  "rev": 4,
  "gid": 1,
  "signature": "ET SCAN NETWORK Incoming Masscan detected",
  "action": "allowed",
  "category": "Detection of a Network Scan"
 },
 "flow_id": 1942230808532336,
 "proto": "006",
 "dest_ip": "172.26.205.101",
 "@version": "1",
 "host": {
  "name": "855db674acac"
 },
 "http": {
  "protocol": "HTTP/1.0",
  "hostname": "172.26.205.101",
  "http_method": "GET",
  "length": 0,
  "xff": "104.152.52.23:44889",
  "url": "/",
  "http_user_agent": "masscan/1.0 (https://github.com/robertdavidgraham/masscan)"
 },
 "dest_port": 80,
 "flow": {
  "pkts_toserver": 4,
```

```
    "start": "2019-06-19T06:42:50.857456+0000",
    "bytes_toclient": 122,
    "bytes_toserver": 786,
    "pkts_toclient": 2
   },
   "timestamp": "2019-06-19T06:42:50.869138+0000"
  },
  "fields": {
   "flow.start": [
    "2019-06-19T06:42:50.857Z"
   ],
   "@timestamp": [
    "2019-06-19T06:42:51.860Z"
   ],
   "timestamp": [
    "2019-06-19T06:42:50.869Z"
   ]
  },
  "highlight": {
   "alert.signature": [
    "ET SCAN NETWORK Incoming @kibana-highlighted-field@Masscan@/kibana-highlighted-field@
detected"
   ]
  },
  "sort": [
   1560926571860
  ]
}
```


4.1.3.6 Monitoring dashboard

The Monitoring dashboard will display all acquired data and generated events in a user-friendly manner in form of alerts, statistics and graphs. It is based on Kibana [26].

Figure 8 and Figure 9 show different graphs that can be checked in the Monitoring dashboard.

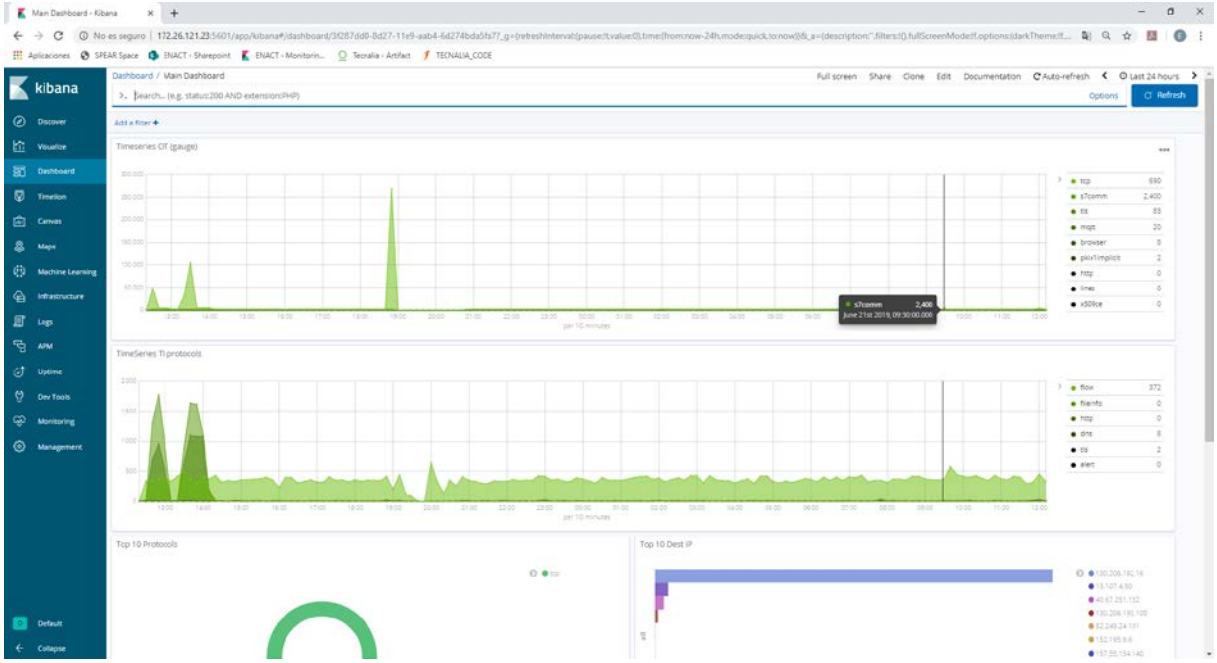

Figure 8. View of the Monitoring Enabler dashboard

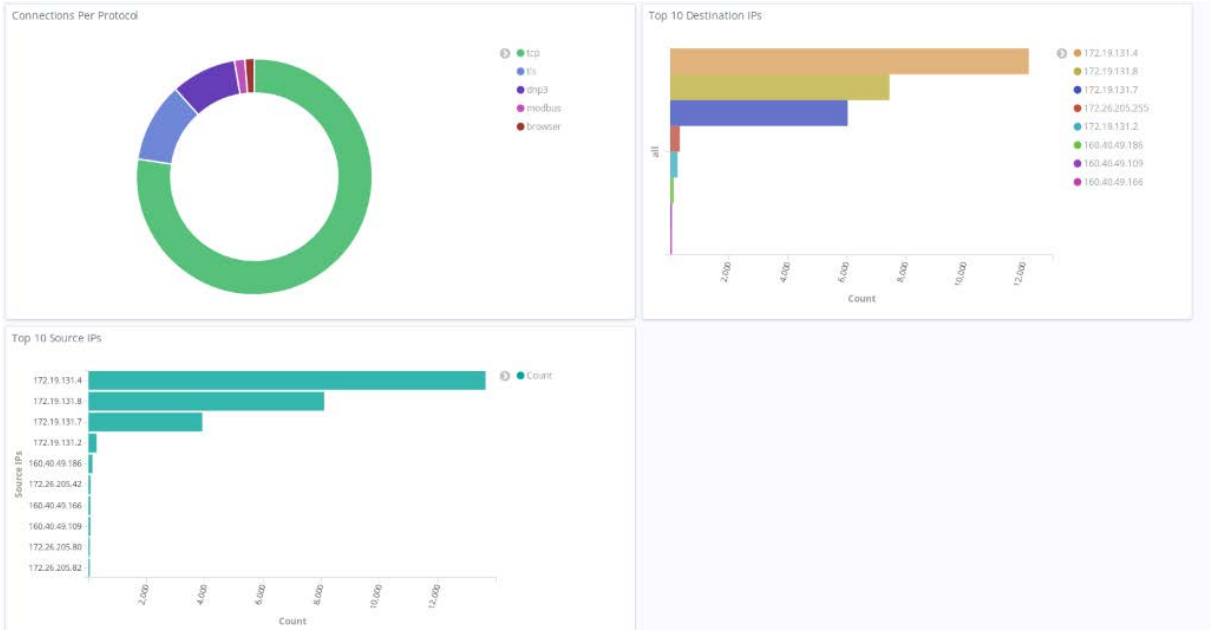

Figure 9. Graphs included in the Monitoring dashboard

Since the development of the Monitoring Enabler is being continuously updated until the final version (M32), the Monitoring dashboard will be enhanced with more statistical and networks graphs. Moreover, it will be further improved with a standardized classification of security events (such as MITRE ATT&CK) and candidate security controls that can mitigate the detected potential threat.

4.1.3.7 Anomaly detection

The first version of the Monitoring Enabler is working without the functionalities of the Anomaly detection component. All the infrastructure of data acquisition, pre-processing and storage is being deployed in the ENACT use cases, therefore the dataset required to train the Anomaly detection is currently being gathered. The final version of the Monitoring Enabler (M32) will include the functionalities of the Anomaly detection component which will correlate information from the multiple distributed probes in the SIS.

4.1.4 Interface

As explained in section 4.1.3.6, the Monitoring Enabler provides a user graphical interface where the operator can check the status of the security of the SIS. The Monitoring Enabler displays the information in form of statistical graphs and alerts. Moreover, the Monitoring enabler can provide all handled data internally to other ENACT enablers with a client for the streaming bus to subscribe to the events and data generated by the Monitoring Enabler [27].

4.1.5 Tutorial

The Monitoring Enabler is delivered as a docker based deployment. The main steps for its deployment are explained next.

Docker needs to be installed in all machines used for deploying the Monitoring Enabler components; Docker installation is well explained in the documentation of Docker [28].

The Docker images of the Monitoring Enabler components will be generated and published in an artefact repository managed by Tecnalia: https://artifact.tecnalia.com/. They are available for authorised internal use of ENACT partners.

Network monitoring agents and NIDS
Instances of the network monitoring agent and NIDS components need to be deployed in each of the networks to be monitored in the SIS. In order to monitor the network traffic that is not being sent to or from the monitoring machine, the network interface of the machine needs to be enabled as promiscuous mode, and on a switched Ethernet network, it also needs to be set up the use of port mirroring.

System monitoring agents
Instances of the system monitoring agents need to be deployed in each of the machines of the SIS to be monitored.

Streaming bus, Data Capturing and Parser, IoT data storage, Monitoring dashboard (Server-side components of the Monitoring Enabler)
The server component parts of the Monitoring Enabler need to be deployed in a machine that at least fulfils the following hardware requirements of the base technologies:

- Streaming bus based on Kafka: HW requirements https://kafka.apache.org/documentation/#hwandos
- IoT data storage based on Elasticsearch: HW requirements https://www.elastic.co/guide/en/elasticsearch/guide/current/hardware.html

App monitoring agent
The app monitoring agent is particular of SMOOL IoT platform and its source code is published in ENACT gitlab: https://gitlab.com/enact/smool_enact.

The installation and usage guides are included there in the README.md file.

This agent execution is closely related to the execution of the SMOOL Platform. The usage guide of SMOOL can be found here: https://bitbucket.org/jasonjxm/smool/wiki/Home

4.2 Security and Privacy Control mechanisms

In this section we explain in detail the mechanisms offered in ENACT for ensuring that the security and privacy properties designed in the Development phase of the DevOps process are actually addressed in the SIS and work properly, i.e. the control mechanisms offered in Operation for the assurance of security and privacy capabilities of the SIS. From all the possible security and privacy aspects in a SIS, the focus of the work in ENACT is the assurance of confidentiality and integrity of the data, be it personal data (privacy) or non-personal data (security).

In the following, first we describe in section 4.2.1 the new mechanism developed for context-aware access control to enable in the IoT systems fine-grained resource access policies associated to context conditions. Second, in section 4.2.2 we describe the controls developed to be used in the IoT Platform layer for IoT systems that use SOFIA middleware. Third, in section 4.2.3 we provide details of the Code Diversifier (DivEnact) developed in ENACT for ensuring SIS adaptability through the update of software versions in SIS components. Finally, section 4.2.4 explains other complementary enforcement mechanism for access control that will be orchestrated in ENACT in cases that the context-awareness is not necessary.

4.2.1 Context-Aware Access Control[13]

4.2.1.1 Purpose of the tool

The Internet of Things links many devices such as sensors, cameras or smartphones to the Internet. These devices have the capacity to act as sensor or actuator in their environment, while the context can continuously change and evolve. The environmental data are considered as dynamic and give crucial information about a context (state of devices, user's behaviour and location, etc.). The traditional mechanisms of access control do not use these contextual data while doing authorization decisions.

The objective of the Context-aware Access Control (CAAC) is to provide mechanisms for controlling the security, privacy and trustworthiness behaviour of smart IoT systems. A specific emphasis is made on confidentiality and integrity of data and services. This includes reaction models and mechanisms that address the adaptation and recovery of the IoT application operation on the basis of the application context, in order to deliver dynamic authorization based on context for both IT and OT (operational technologies) domains.

Evidian Web Access Manager (WAM) provides security features for identity management and access control based on the protocols OAuth2 and OpenID Connect (OIDC). The **Context-Aware Access Control tool** is an evolution of the authentication and authorization mechanisms provided by WAM intended for the Internet of Things.

Due to the dynamic of the data concerning the environment or a person, the contextual information must be used to manage and adjust these security mechanisms. In this objective, the protocol OpenID Connect is extended to also consider dynamic attributes on the user and thus apply security rules.

4.2.1.2 Architecture

The Context-aware Access Control tool provides an Authorization mechanism that issues access tokens to the connected objects after successfully authenticating their owner and obtaining authorization. This Authorization mechanism uses the OAuth2 protocol, which provides authorization delegation mechanism. Following this protocol, an object can access a backend API by using an access token containing the list of claims (i.e. user's attributes. Ex: user name, email address, etc.) and scopes (read-only, read/write) that an authenticated user has consented for this object to access. An Access token contains an authentication proof and the list of consented scopes and claims to access the asked resource.

This Authorization mechanism may be coupled with contextual information to adapt the access authorizations according to them (for example to make certain information more widely available in some urgent case).

The Context-aware Access Control tool provides access tokens that allow a Reverse Proxy working as an API Gateway to control the access to applications and APIs. The scopes and claims contained in the access tokens are used to restrict accesses to the backend server APIs to a consented set of resources.

The Authorization mechanism could be coupled to a multi-level, multi-factor Authentication Server that provides strong authentications mechanisms to the users. This mechanism mitigates the level of authentication required depending on the user's environment context and an external context. The risk is a value computed either statically, depending on a defined configuration, or dynamically by using a REST API to dialog with an external decision engine. The input used to compute the risk is the user's session context, which contains the browser DNA, the service the user wants to access and the configured trust level of this service, the access time and the trust planning associated to the service, and the IP address and its geolocation. Depending on the evaluated risk of the user's session, the level of the required authentication will be leveled up, or, if the risk is too high, the connection will be refused.

[13] EVIDIAN Proprietary software. A first release of the Interface documentation is available, and the API can be exposed for testing and validation purpose.

The Figure 10 shows a use case example of the Context-aware Access Control tool in which various actors interact with WAM:

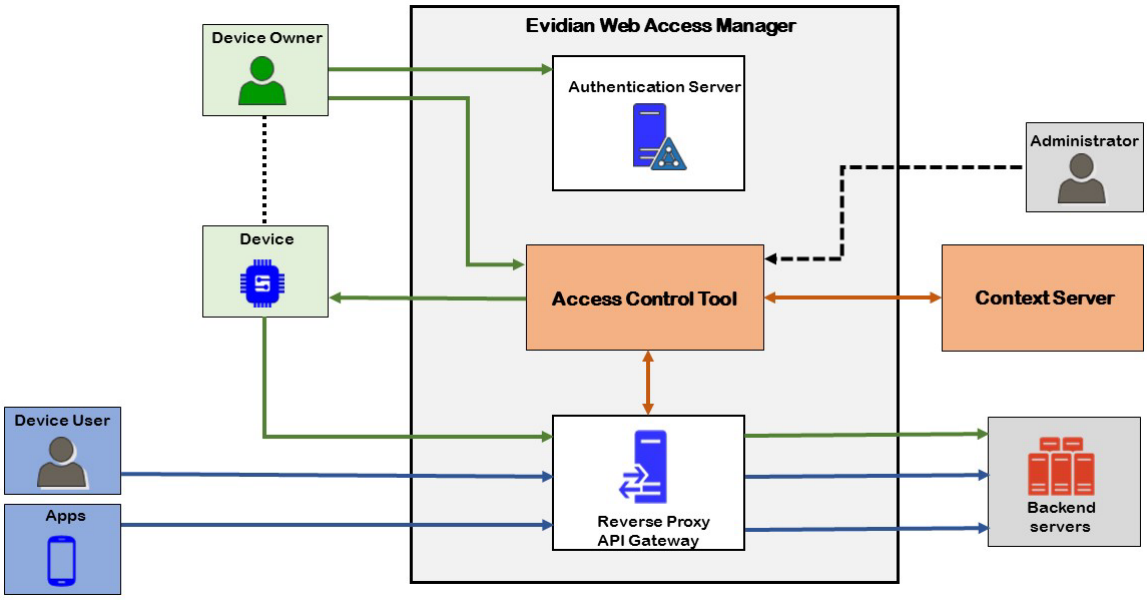

Figure 10. Use case example of the Context-aware Access Control

During the enrolment phase, a connected object is associated with a user; then the connected object can push data to a backend server in a controlled way by using the access token it received from the Access Control Tool. The backend server stores this data and can display it within an application. The authorized users and applications can retrieve the data from the backend server. WAM plays a pivotal role between all these exchanges by making authorization decisions depending on the context.

The Context-aware Access Control tool built on the OpenID Connect protocol includes a scopes system. The scopes are used by an application to authorize access to a user information, like name and email. In the OpenID Connect protocol, the scopes system only returns a set of static user attributes. The Context-aware Access Control tool extends the protocol to also consider dynamic attributes which provides contextual data on the user and his devices. That way, the access control can be adapted depending on the context which can be continuously evolving, in order to make the access rights more secure and efficient in function of the current environment.

The Access Control Tool directly communicates with a Context Server to make dynamic access controls based on the context information during the authorization phase. For example, it can reject the authorization if the access token is valid while other context information does not respect the authorization policy.

The authorization policy is a set of rules that define whether a user or device must be permitted or denied accessing to a backend server. An administrator can control this adjustment and create special authorization rules based on the context data provided.

In this architecture, two components are providing the Context-aware Access Control mechanisms:

- **The Context server**

The Context Server exposes a REST API that provides contextual data on the user and his devices. These data are dynamic attributes and come from other external sources (sensors, other applications, etc.).

WAM looks after the Context Server to link in its repository the user's profile and the user's dynamic information. All information about the user's profile, dynamic attributes and devices associated with the user are saved in the WAM repository.

In the example used in the rest of this chapter, the user's contextual data is presented as a "risk value". In this case, the Context Server publishes a certain "risk value" about a user. This risk value is an arbitrary value that informs on the current state of the user. This information is dynamic and changes periodically. WAM updates the dynamic attributes related to the user and his devices on WAM repository by polling the Context Server when this information is needed.

Example of contextual data for the user werner:

```
risk {
  "user": "werner",
  "serialNumber": "XXX",
  "threshold": 5,
  "value": 2
}
```

In this example, according to the "risk value", the request of the device to access the backend server can be rejected if this value reaches a critical threshold, depending on the rules created by the administrator.

- **The Access Control Tool**

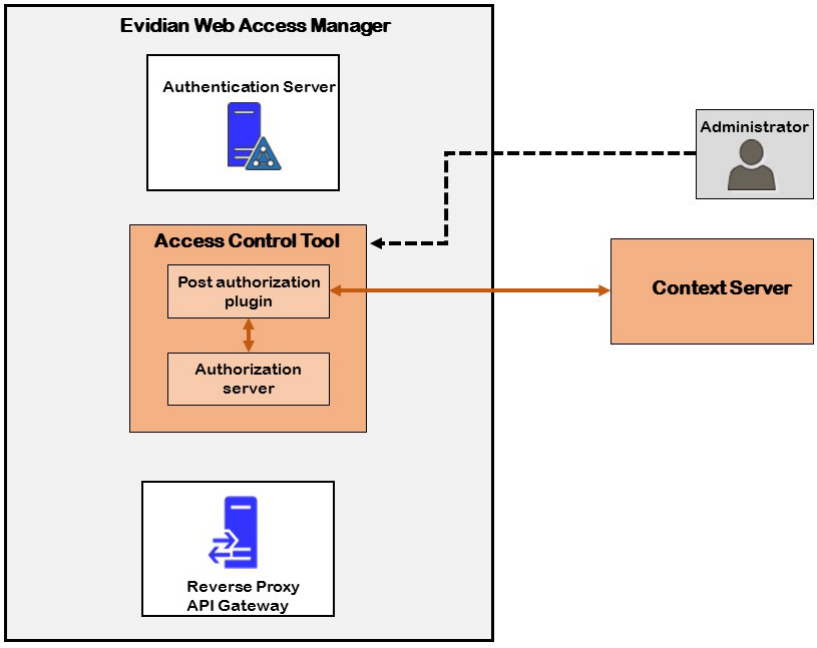

Figure 11. Access Control tool

The Access Control Tool of WAM depicted in Figure 11 is composed of an Authorization server associated to a Post authorization plugin, to add more controls during the authorization phase. Its purpose is to check if the request is authenticated and is authorized to access the backend server.

Indeed, each time a device sends a request to a backend server, WAM can check the dynamic scopes about the user associated to the device that performs the request and realize special actions according to this information like blocking the request or limiting the accessible scopes.

The Post authorization plugin extends the basic authorization phase and is entirely customizable. Any operation can be executed during the authorization phase, including calling external programs, and in particular the Context Server. The Post authorization plugin can create injection variables that can be reused and injected in the initial request sent to the backend server. This is detailed in section 4.2.1.4.

The Administrator can configure the Access Control Tool to adjust the authorization security rules according to the dynamic attributes.

4.2.1.3 Detailed design

The **Context-aware Access Control** mechanisms work in several steps:

- **Device enrolment**

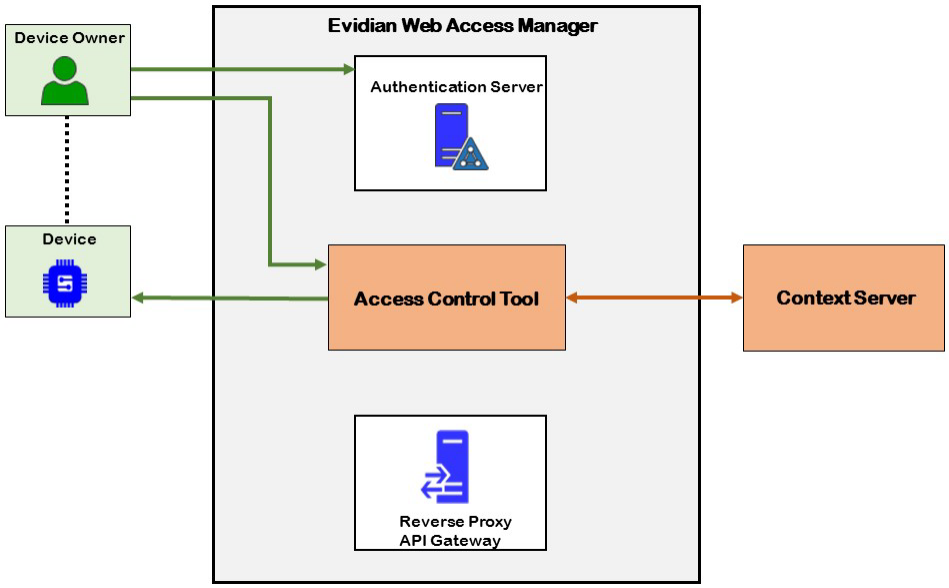

Figure 12. Device enrolment

The device enrolment step depicted in Figure 12 allows a device to be associated with the identity of its owner. The Access Control tool leverages on the OAuth 2.0 Device Flow protocol to achieve this.

The only requirements to use this flow are that the device is connected to the Internet and able to make outbound HTTPS requests, and that it is able to display or otherwise communicate a URI and code sequence to the user, and that the user (device owner) has a secondary device (e.g., personal computer or smartphone) from which to process the request. There is no requirement for two-way communication between the OAuth client (i.e. the connected device) and the end user's user-agent, enabling a broad range of use-cases.

At the end of the enrolment phase, the device receives an access token. The device has now access to the device owner profile that includes static attributes (username, email, etc.) but also dynamic attributes (contextual information on the user).

The sequence diagram for this device enrolment step is described in Figure 13

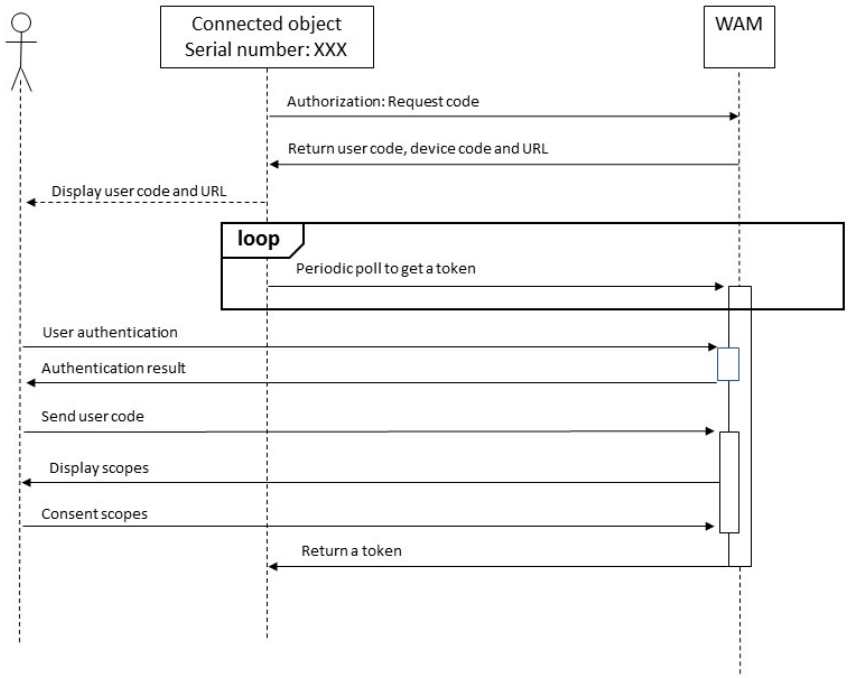

Figure 13. Device enrolment sequence diagram

- **Device access to the authorized resources**

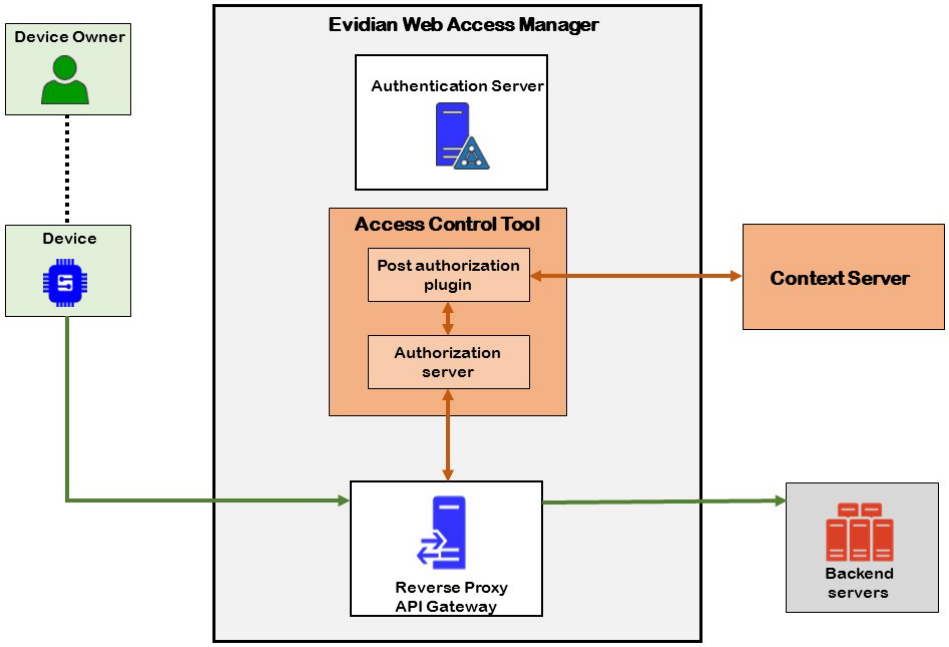

Figure 14. Device access control

The device access control mechanism is depicted in Figure 14. The device makes a request to the backend server by using its access token. WAM uses this token to authenticate the device. After the authentication, WAM verifies if the device can access to the authorized resources and checks the device owner's dynamic attributes. The role of WAM is here to control the access to the backend servers according to the device owner's attributes.

- **User's data access**

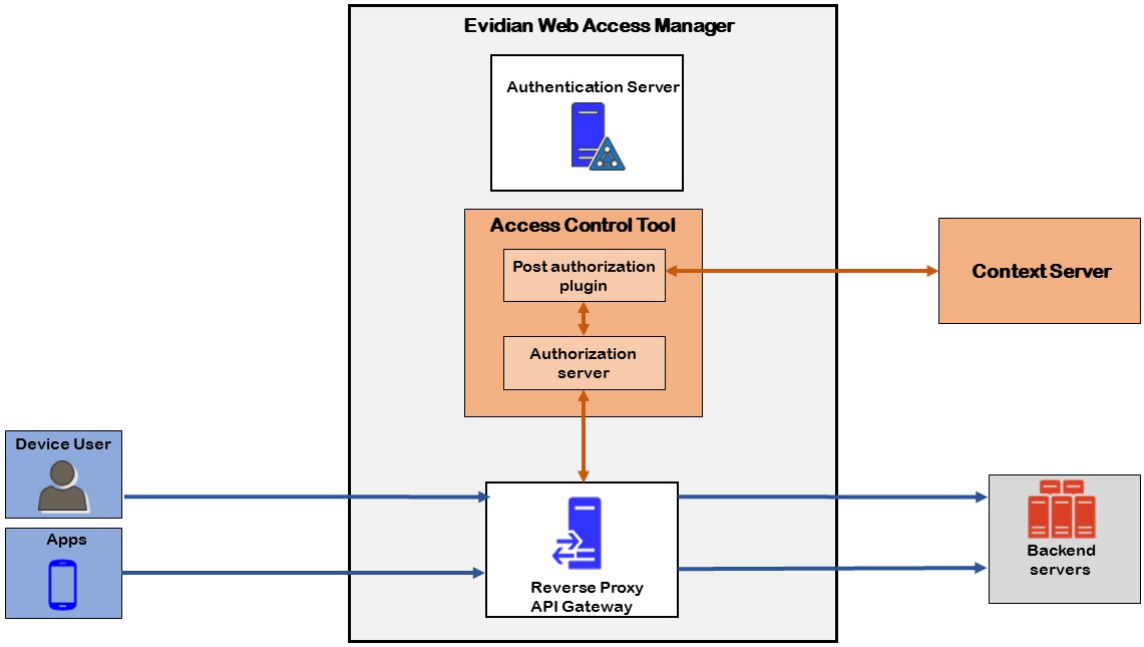

Figure 15. Users data access control

As depicted in Figure 15, the device user or other applications can retrieve users data on the backend server if they are authenticated and authorized. The Access Control tool adjusts the authorization according to the contextual data concerning the device and the user.

4.2.1.4 Use case integration manual

The Context-aware Access Control tool can work in various ways when integrated in use cases:

- **Context-aware Access Control with WAM used as reverse-proxy**

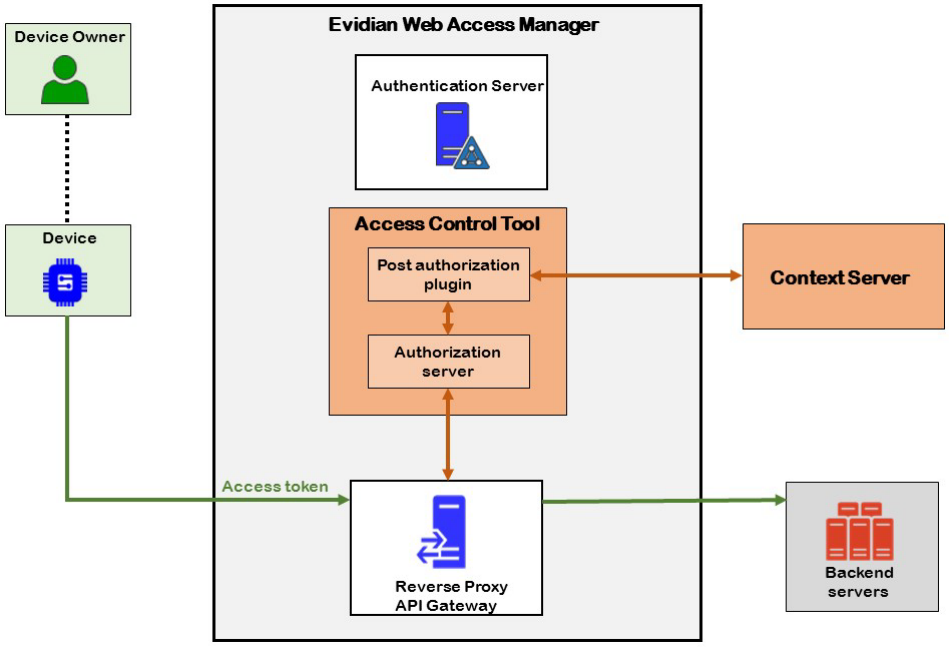

Figure 16. Context-aware Access control with WAM as reverse-proxy

In this case depicted in Figure 16, WAM is used as a Reverse proxy to protect the backend servers. Additionally, WAM checks the token of the incoming request to verify if the device is authorized to access the backend server. If this is the case, WAM injects in the header of the initial request the consent scopes of the device owner. This injection does not modify the request and the scopes injected contain some information about the device owner (username, email, etc..) and some contextual information. It allows the backend server to make the link between the requesting device and the user associated with it, and to get additional contextual information that it can take into account.

Let's take an example of a connected arm tensiometer that needs to push some tension data to a backend server. Here the role of the backend server is to store the data received from the tensiometers. To do this, the device will make an HTTP POST request with its access token to the reverse proxy that protects the backend server. WAM checks the validity of the token and it verifies if all the attributes (static and dynamic) of the user associated with the device respect the authorization policy. This verification of the dynamic attributes is the context verification done during the authorization step. For instance, the contextual dynamic information may be about the position of the user (standing or lying down), or the temperature of the room, while the arm tensiometer is active. If the token is valid and the user's attributes respect the authorization policy, WAM injects the consent scopes in the header of the original request. This consent scopes injected can be static like the identity of the user (username, email) or dynamic like the "risk value" related to this specific user. The backend server can now store the tension data associated to the user and the related context.

The sequence diagram for this working mode is described in Figure 17).

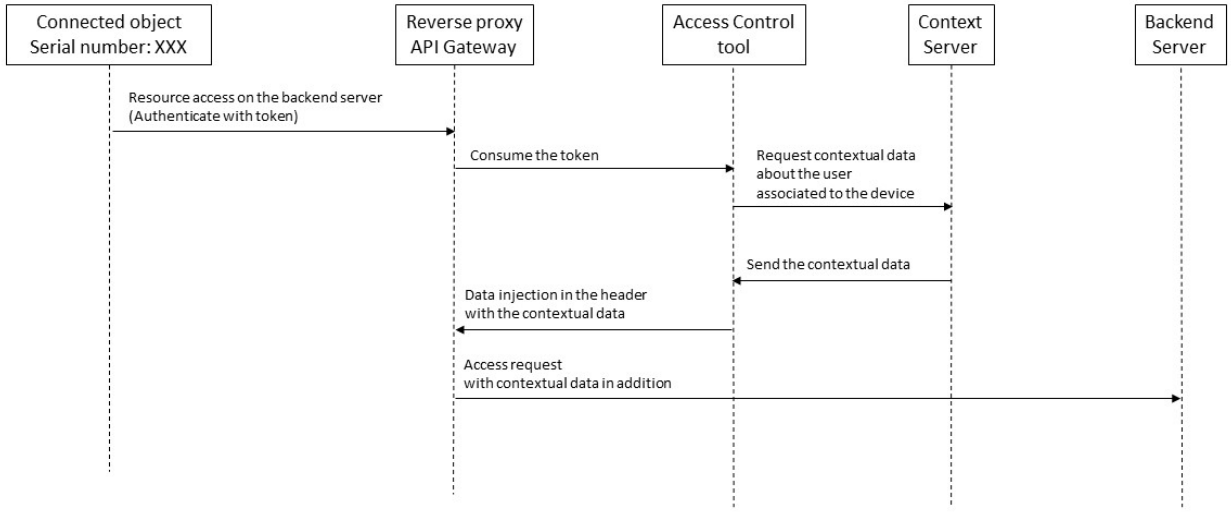

Figure 17. Context-aware Access control with WAM as reverse-proxy – Sequence diagram

This sequence diagram explains the mechanism of Context-aware Access Control with WAM acting as a reverse-proxy. Each time a device performs a request to a backend server with its access token, WAM consumes the token to verify the identity of the user associated to this device. After this, WAM retrieves some contextual data about the device and the user from the Context Server. Then these data are injected in the header of the initial request and the request is sent to the protected backend server.

- **Context-aware Access Control with WAM not used as reverse-proxy**

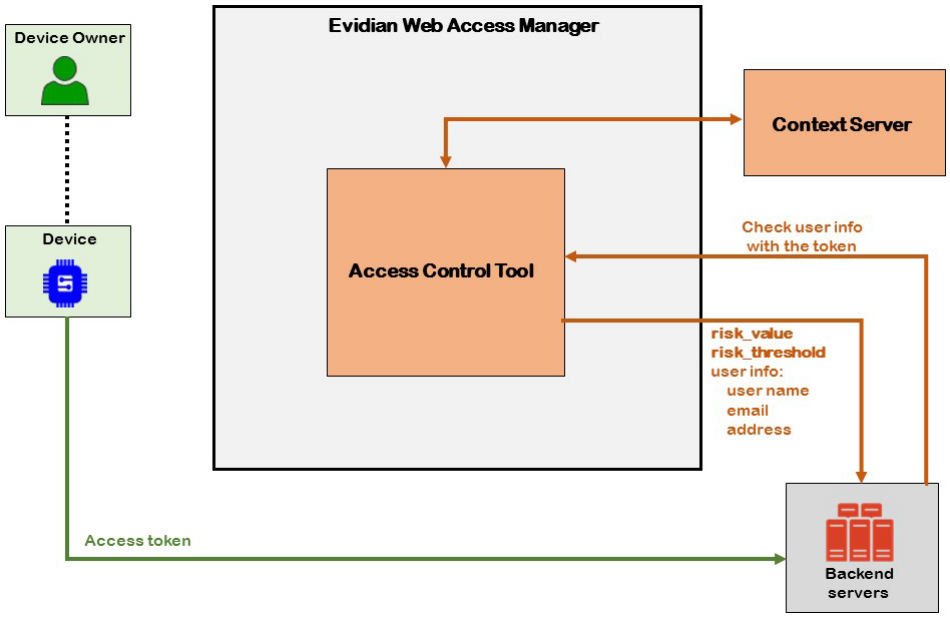

Figure 18. Context-aware Access control with WAM without reverse-proxy

In this case depicted in Figure 18, WAM is no longer used as a Reverse proxy. WAM only handles the access control part by checking if the token sent to a backend server is valid and if all the user attributes (static and dynamic) associated with the device respect the authorization policy. If this is the case, WAM responds with the consent scopes of the device owner. These scopes contain some information about the user (username, email, etc.) and some contextual information.

Taking our previous example, the connected arm tensiometer needs to push data to the backend server. To do this, it directly makes an HTTP POST request with its access token to the backend server. The backend server must validate this request by asking some user information in respect with the token to the Access Control tool. The backend server then receives some user information and can now match the received data with a user.

In the case where the authorization policy is not respected, the Access Control tool will inform the backend server that it has to reject the request made by the device, as depicted in Figure 19:

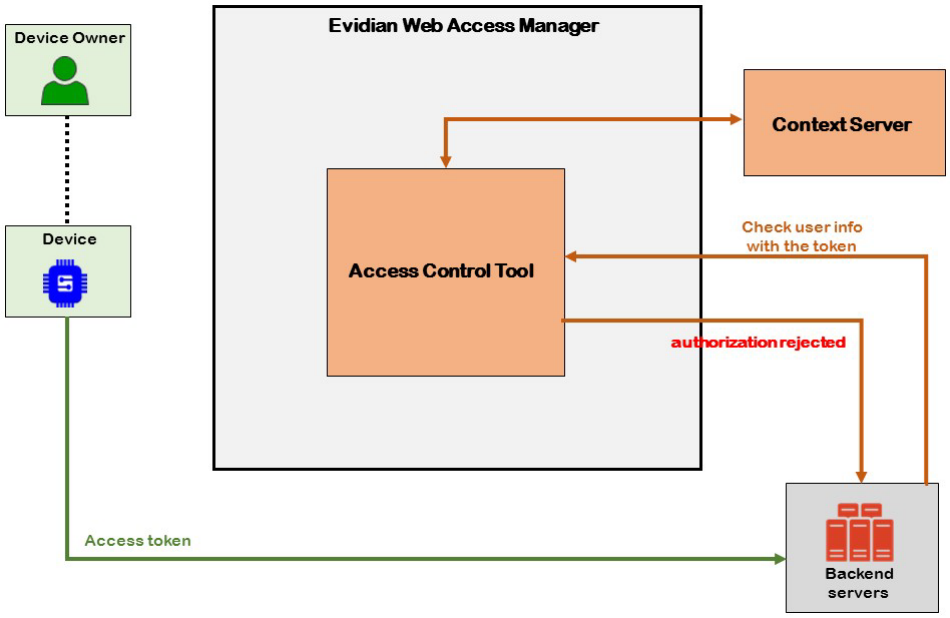

Figure 19. Context-aware Access control with WAM without reverse-proxy – Access denied

The sequence diagram for this working mode is described in Figure 20:

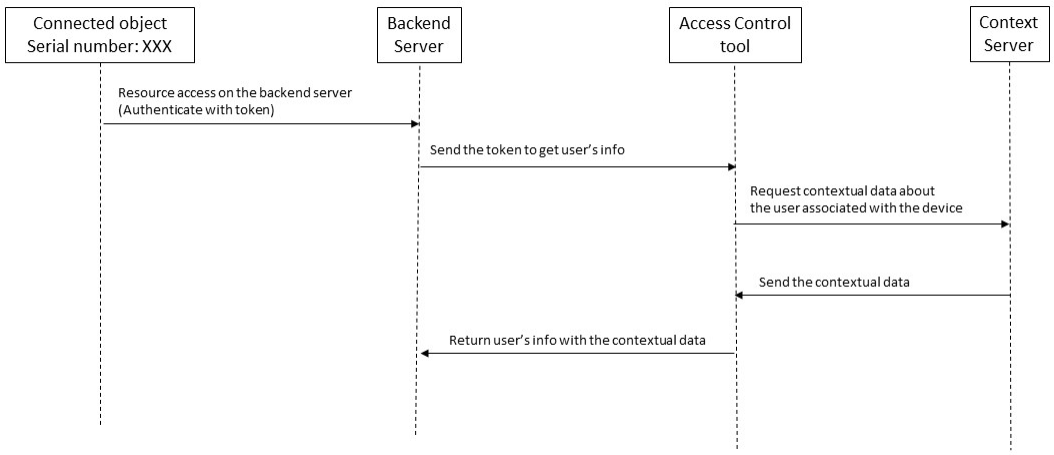

Figure 20. Context-aware Access control with WAM without reverse-proxy – Sequence diagram

The device uses its token to access the backend server (for example to push some data). The backend server checks the validity of the token and retrieves the device owner's consent scopes for this token by calling the userinfo endpoint of WAM (the userinfo endpoint from the Access Control tool API consumes a token to retrieve information on the user). WAM returns the user information (for instance: username, email, address) and dynamic contextual data (for instance: risk value, risk threshold). The backend applications can use this additional information to perform special actions.

- **Retrieve the contextual data from the device**

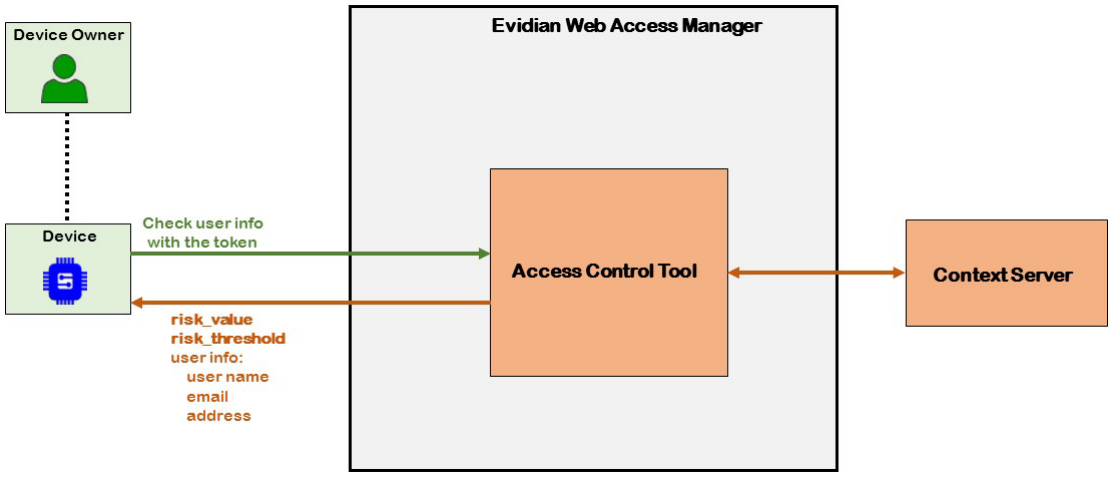

Figure 21. Retrieve the contextual data from the device

This mechanism depicted in Figure 21 enables the device to get some information on the user associated with it and some contextual data.

For example, the arm tensiometer needs to display on a little screen the name of the user and a dynamic data like the risk value. The device can periodically perform request to the Access Control tool to retrieve this data and display it on the screen.

In the following example, the device performs a request to the user info endpoint of the Access Control tool with its access token. Then, the Access Control tool responds with the user information and the contextual data on the user associated with the device.

Request to get information on the user and contextual data:

```
{
POST /form/oidc/userinfo HTTP/1.1
Host: stephane.test-pxp.frec.bull.fr:8081
Content-Type: application/json
Authorization: Bearer eyJraWQiOiJyc2EyNTYiLCJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJld29pYzNW
aUlpQTZJQ0p6YldsMGFFQkNkV2xzZEMxcGJpQlZjMlZ5SjNNZ1JHbHlaV04wYjNKNUlpd0tJbTVoYldVaUl
Eb2dJa3B2YUc0Z1UyMXBkR2dpTEFvaVoybDJaVzVmYm1GdFpTSWdPaUFpU205b2JpSXNDaUptWVcxcGJIbG
ZibUZ0WlNJZ09pQWlVMjFwZEdnaUxBb2ljSEpsWm1WeWNtVmtYM1Z6WlhKdVlXMWxJaUE2SUNKemJXbDBhQ
0lzQ2lKblpXNWtaWElpSURvZ0ltMWhiR1VpTEFvaWFYTmZiMmxrWTE5aFpHMXBibWx6ZEhKaGRHOXlJaUE2
SUdaaGJITmxDbjA9IiwiYXpwIjoiMjZkZDZjYzQtOWE2MC00OGM1LThmNmMtZTQ2MGExNjExNjFiIiwiaXN
zIjoiaHR0cDpcL1wvc3RlcGhhbmUudGVzdC1weHAuZnJlYy5idWxsLmZyOjgwODFcL29pZGMtcHJvdmlkZX
JcLyIsImV4cCI6MTU1MzY4MjAwNywiaWF0IjoxNTUzNjc4NDA3LCJqdGkiOiI1NGIwMzUzNC1lYzk3LTQ1N
2MtYTVmZC00NTc4NjAxNDI1ODQifQ.gWv7jHGaYT8bPV46vFBFuvlRVZNa5z0ud5Qwg03NOt1RUc4LeSUTG
P-ZoasRVseXk46Y6RmIj_HT_x1c7QxLgseIQzgt31kOPFsuqU1hjpUHp5pjs14XM1NlATo_VCJnnMUVv27U
F5gGFCSeOybol1OQGttglbWhmzvYBxwBlcT7msJtq6ODkRGTVu1KMAXBmuoJBMpcqi8XFlgDYdwGt8qbSD2
V2v1Ia8MV-WnsLq_eaf81w_0g7PIdvHXRRn7Zfn5xH23v11Pl9vEfWC9kOZpM_ss_KoGF5QrxdTqUzaEaK_
jmCFQ_XdOpsLXYqCFP5P3zYxu_tM4xKNmOoj3TnQ
}
```

Response:

```
Risk {
"threshold": 5,
"value": 2
Userinfo:
   {
    "sub": "werner",
    "email": "werner@evidian.com",
```

```
    "email_verified": false,
    "address": {},
   }
}
```

4.2.2 Security and Privacy controls in IoT platform

As part of the security and privacy controls developed in ENACT, the Control Manager will be able to manage the behaviour of the IoT Platform SMOOL (based on SOFIA technology [15]) which is the platform used in the Smart building use case in ENACT. This will allow to enable the control of the communications between the "things" in the IoT environment by profiting from the intermediary nature of the IoT Platform in such communications.

To this end, SMOOL has been extended with a number of features oriented towards the full control of security and privacy of the data exchanged in the environment. Particularly, the features developed in the current prototype to embed control capabilities in SMOOL are the following:

4.2.2.1 Security extensions to SOFIA middleware ontology

As described in deliverable D4.1, SOFIA is a semantic middleware that enables the communication between data publishers and subscribers on the basis of semantic concepts. The middleware relies in two main elements:

- *Knowledge Processors (KP)* which are end points of the smart IoT applications where the logic of the IoT application is implemented. These end-points produce and/or consume data to fulfil their tasks. An example of KP is a temperature sensor (of particular type and of particular vendor), a lightning actuator, a web service providing data, etc.
- *Semantic Information Broker (SIB)* which manages the intermediation between the KPs. To this aim, it enables sharing ontology-based semantic information between KPs and acts as gateway that controls the communication technology/network (TCP/IP stack, Bluetooth, etc) of messages transmitted between KPs.

The SMOOL middleware is the open source version of SOFIA that is used in the Smart building use case of ENACT. This middleware has been enhanced in ENACT to be able to control some security and privacy capabilities of the KPs intercommunicating through SMOOL. Particularly, the ontology existing in the baseline SOFIA middleware has been extended to include semantic concepts capturing security metadata of the communications and capabilities of the "things" exchanging data (see Figure 22). These concepts include notions related to subscriber's and/or publisher's identity authentication, subscriber and/or publisher authorization, data confidentiality, data integrity and non-repudiation of the data origin and/or integrity.

By means of these extensions in the ontology, the message format in SMOOL protocol has been enriched so as particular security attributes can be monitored in the communications. The SMOOL messages in Source Service Access Point (SSAP) interoperability protocol are transmitted in raw TCP (not Transport Layer Security -TLS- protocol), and therefore the security metadata could be used to implement within the middleware the security layer on top of raw TCP.

4.2.2.2 Monitoring mechanisms in SMOOL

Within IoT environments or systems that use IoT platforms based on semantic middleware such as SOFIA, the communications between the things can be monitored by means of exploiting the communication gateway and interoperability features of the middleware. To this aim, SMOOL version of SOFIA has been extended in ENACT to create a Security KP specialised client of SMOOL which is able to interpret SMOOL protocol messages. The Security KP acts as an application agent (App agent in Figure 5) and log all the needed information from the communications which will be analysed by the Control Manager.

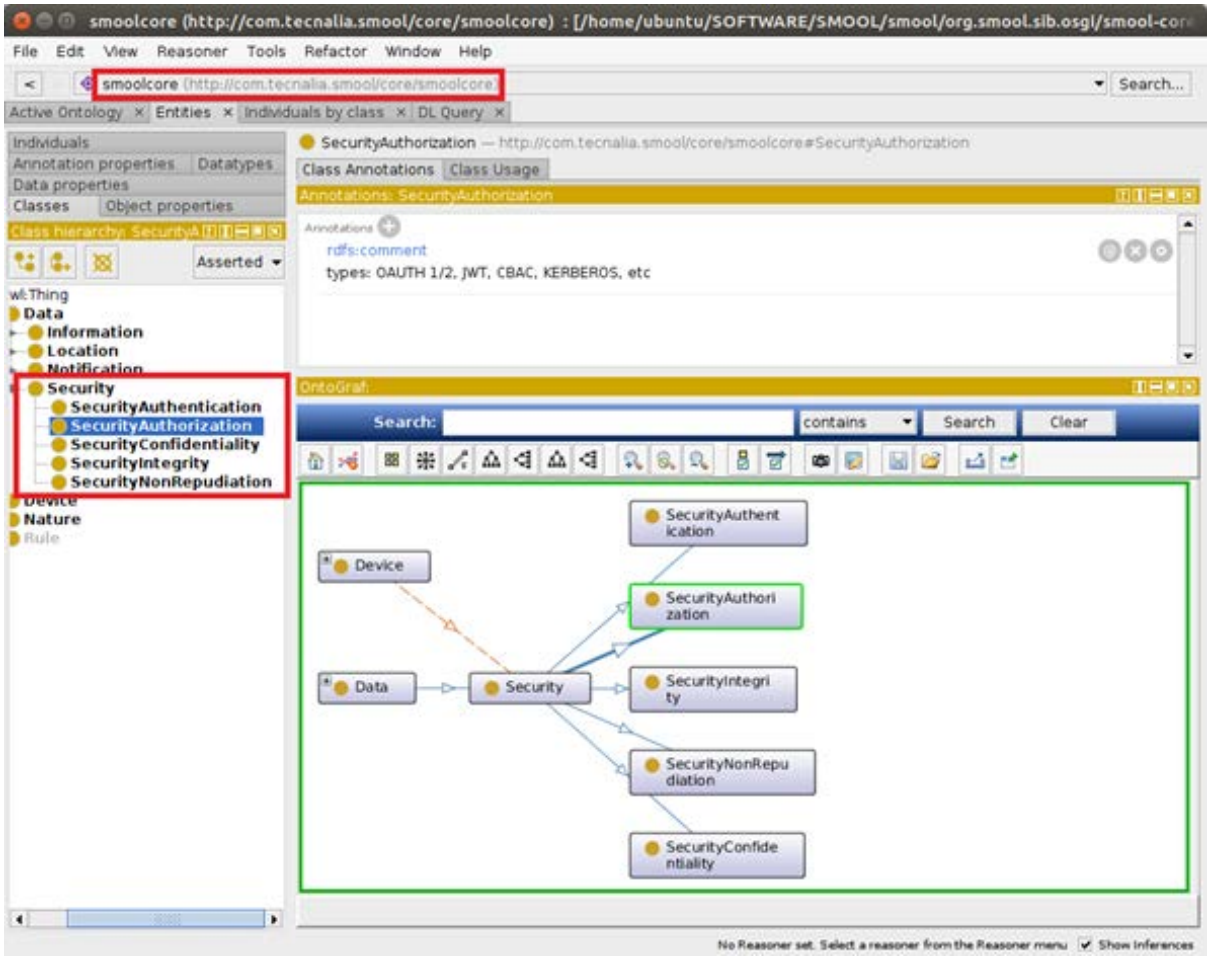

Figure 22. Security metadata extensions in SMOOL ontology

4.2.2.3 Control mechanisms in SMOOL

Within the Security and Privacy Monitoring and Control Enabler architecture (Figure 5) an **app agent** has been included so as to be able to control the application layer messages confidentiality and integrity. This app agent has been implemented as a special mechanism able to work with IoT systems that integrate SMOOL IoT Platform as the communication middleware between the things and/or between the things and the system services (potentially in the Cloud).

The app agent developed in ENACT is a special SMOOL Knowledge Processor in charge of intercepting the communications through SMOOL for their further security analysis. This SMOOL KP has been named SMOOL Security KP and it will act as application layer agent designed to collaborate with the Control Manager in the Security & Privacy Monitoring and Control Enabler. The new Security KP is able to send application data and metadata to the Control Manager, which is the module taking the decisions and verifying whether specific detection rules apply over the data and metadata.

This way, SMOOL Security KP can request the Control Manager to check specific anomaly detection rules over the attributes of the IoT clients or the message itself. In case the anomaly detection rule verifies to true, the Control Manager would invoke the services in the SMOOL Server to control communications, i.e. request SMOOL Server to block the potential attacker IoT client or perform commands to start collecting additional data on the attacker prior to blocking it, etc. For example, the Control Manager can request that communications for a particular publisher or subscriber are blocked due to obsolete timestamps, unknown vendor, wrong encryption algorithm, obsolete certificate, etc.

The control mechanisms executed by SMOOL are complementary to other controls managed by the Control Manager in other layers, e.g. the context-aware access control by the CAAC tool in ENACT. While the controls in SMOOL are based on the analysis of application layer message content and they will only be available for IoT scenarios where SMOOL semantic platform is used, other controls will be adopted in other scenarios. For example, the CCAC tool will be used in scenarios where access policies to SIS services, resources or "things" will need to be adapted to context condition changes. In some cases, both SMOOL controls and CAAC could be used together as a double-layer protection mechanism.
Note that the communication between the Security KP and the Control Manager is secured by the use of HTTPS and an authorization header only known by both of them.

In the example message described in Figure 23 a SMOOL client (data producer) is sending encrypted metadata (encrypted with ChaCha20 algorithm) along with temperature information. The Security KP supports also the transmission of authorization data such as tokens (e.g., Json Web Token (JWT), shown as commented code in green in the Figure 23). In the lower part of the figure, the SMOOL SSAP message is shown containing the security data transmission.

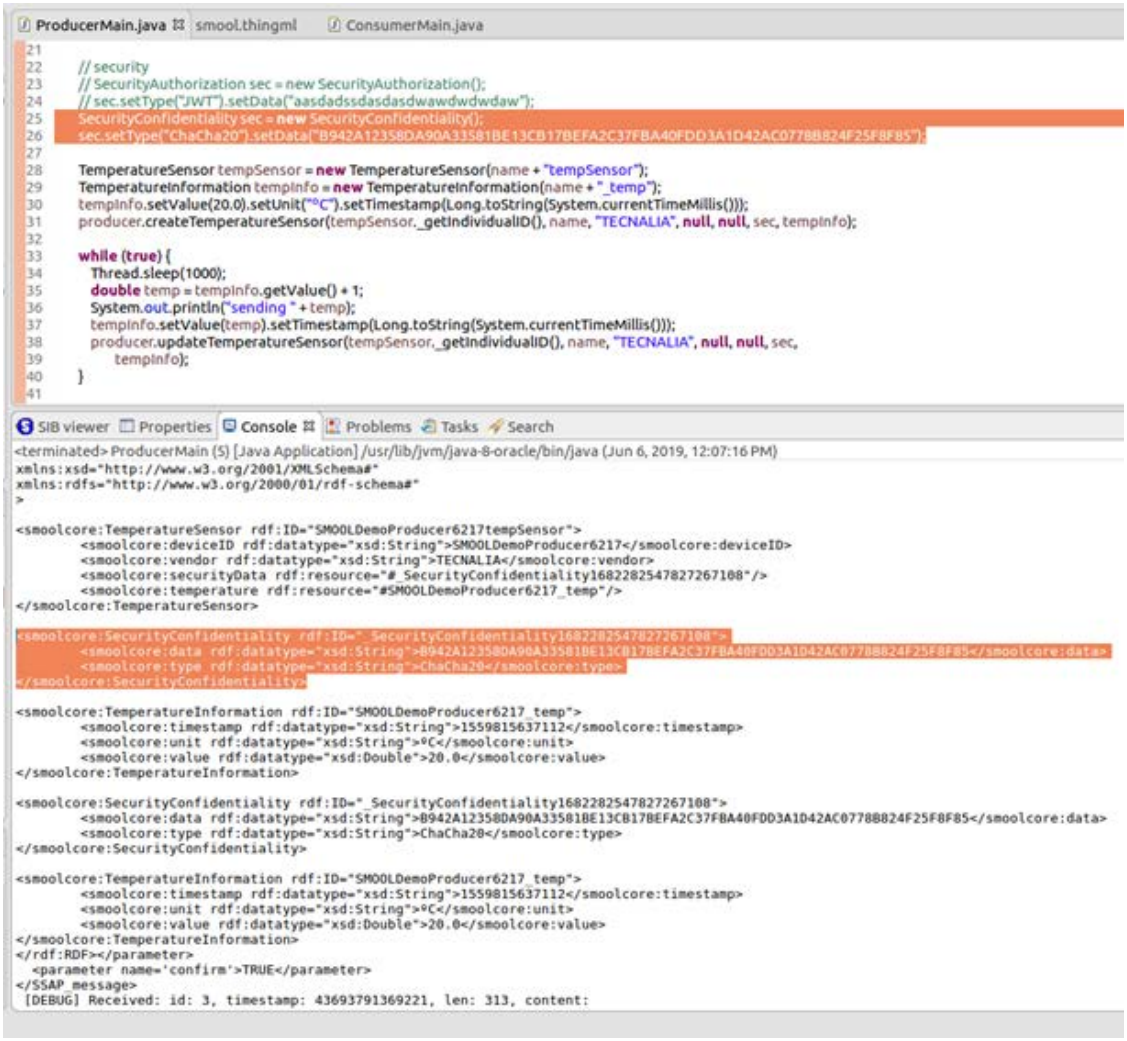

Figure 23. New message format in IoT Producers and Consumers to include security metadata

4.2.3 Code Diversifier

4.2.3.1 Purpose

In commercial IoT applications, a vendor usually serves many customers simultaneously, each of which hosts an IoT system (or sub-system), containing a number of sensors, one or multiple gateways, and software components deployed on top of them. Taking the e-Health use case by TellU as an example, the vendor, TellU, has hundreds of patients as its customers, and each customer has a TellU gateway, connected to several healthcare sensors. The sensors collect healthcare data from the patient or the physical environment and send the data to the vendors via the gateway. TellU operates a set of application-level services running either on the gateway or in the cloud, and these services analyse the data and drive interactions with the patient or the healthcare staff.

The subsystems deployed for different customers are initially identical, but during the lifecycle of the entire IoT application system they will become more and more different from each other. Such diversity may come from different sources:

- Hardware diversity: different customers may utilize different sensors, due to different functional requirements or subscriptions.
- Hardware evolution: new customers may be assigned later versions of gateways or sensors, whereas the vendors may keep older versions of hardware deployed for existing customers.
- Software versions: the vendors may deploy different versions of software components on the gateways.
- Alternative software or libraries: some functions can be implemented by different software components, and different customers may be provisioned with different ones.
- Diverse configurations: the software components may be configured by different parameters or with different topologies between each other.

Such diversities may emerge naturally during the operation of the system or be injected by the vendors. The emergent diversity may be caused by the difference of custom requirements, the different times of the customer's provisioning and upgrading. On the other hand, the vendors may intentionally introduce or inject diversity into the system, e.g., to test variants on divided groups of customers (A/B testing), or to increase resilience of the whole system. To inject diversity, vendors can use the code and architecture diversification tools as described in Section 3.

The ENACT diversification controller at runtime, with the tool name DivEnact[14], aims at managing a large and dynamic fleet of sub-systems with emerging and injected diversity, in order to enhance the robustness and resilience of the entire system. In short, it monitors and records the diversity among subsystems, manages the lifecycles of these subsystems, and controls the upgrading, deployment and modification of software components on these subsystems.

DivEnact is an extension to GeneSIS, the IoT deployment tool as described in Deliverable 2.2, moving from the deployment of one IoT system to the deployment of a fleet of similar, or functionally identical, IoT systems. In short, DivEnact decides what software components should be deployed to each sub system, and when to deploy or update them. The actual deployment action on each sub system is the responsibility of the IoT deployment engine, such as GeneSIS. In the first prototype, DivEnact utilizes the built-in deployment engine of Azure IoT Hub (which is the underlying platform used by DivEnact). The Azure deployment engine has the same objective as GeneSIS, but only support the deployment into the Edge level (which technically means that the device should run a fully functional Linux distribution). In the next step, we will replace this built-in engine by GeneSIS, in order to achieve the full-functional deployment, covering the deployment of code into IoT devices, and the deployment of cloud services.

[14] Source code repository: *https://github.com/SINTEF-9012/divenact*

4.2.3.2 Architecture

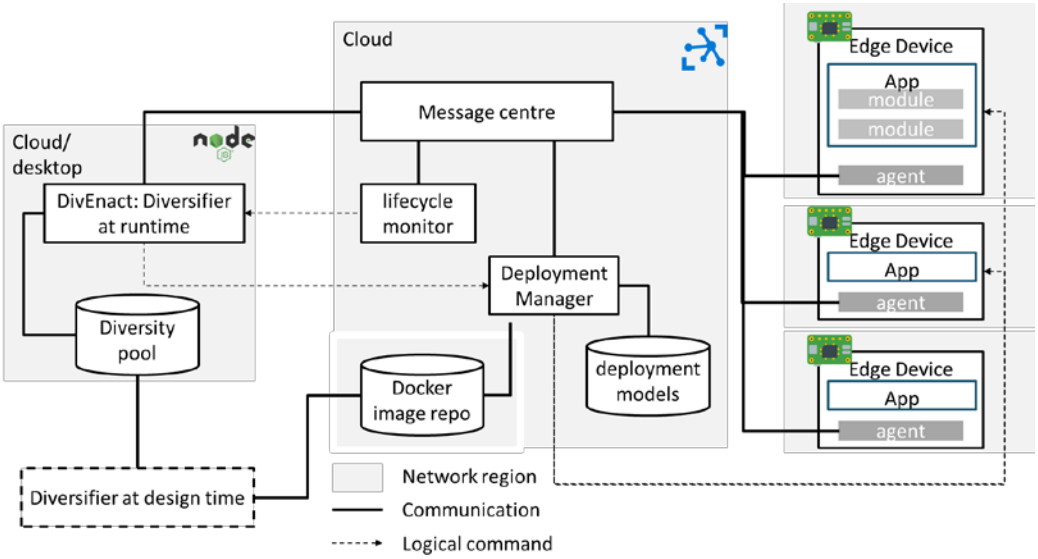

Figure 24. Architecture of ENACT Diversifier at runtime (DivEnact)

Figure 24 shows the architecture of DivEnact (the ENACT diversification controller at runtime). In summary, DivEnact (on the left of the diagram) is a back-end service running on a desktop or a cloud virtual machine, and it controls a fleet of edge devices, shown on the right. Each edge device may further be coupled with a number of IoT devices, i.e., sensors or actuators. We omit the devices in this drawing, since they are not the main focus of this architecture. An edge device is also deployed with one or multiple applications, each of which contains several modules. The applications collect data from the sensors and send it to the application services, usually deployed in the cloud. Again, we omit the application services in the drawing, as they are not the focus of the current prototype. An IoT application vendor develops and operates the application services, and also maintains a fleet of such edge devices for their clients. As we have discussed in the previous section, these devices will be diverse – i.e., become different from each other during the lifecycle. From the software point of view, such diversity is realized by different modules deployed into those devices. The main feature of the DivEnact tool is to manage diverse deployments of application modules on edge devices. Here, each edge device represents an IoT system deployed for a particular customer. Behind each edge device in the architecture, there is a system with more IoT and Edge devices, and the distribution of software components in these devices will be handled by GeneSIS, in the next step of DivEnact.

The primary principle of DivEnact's architecture is loose coupling, i.e., the **separation between the diversity controller and the edge devices.** In particular, the diversity controller never interacts directly with the edge devices. The reason is twofold. On the one hand, the edge devices are deployed remotely, scattered across different physical locations, in a possibly unstable environment in terms of power and network. In the context of a large-scale system, it is very likely that when the operators work through the backend service, an edge device is not connected, but the change they make to its deployment must be enacted the first time when the device is connected again.

The main design following this principle is to have a central hub between the backend and the edge devices, as shown in the middle of Figure 24. DivEnact is designed to be compatible with multiple solutions for such hubs, and as the first prototype, we currently support Azure IoT hub. The central hub has a message centre, which allows both the backend service and all the devices to communicate via APIs and async messages. The devices send heart beat messages to report their current statuses, which are recorded by the lifecycle manager. The deployment manager checks the current deployment of the devices and modifies their deployment configuration when necessary based on a set of predefined

deployment models. Each deployment model contains a condition, which defines what devices are affected by this deployment. When a new deployment model is created, or an existing one is updated, the deployment engine automatically updates the modules of all the devices that satisfy the target condition of this deployment model.

The DivEnact service interacts with the deployment manager via the message centre. In particular, it generates deployment models based on a pool of candidate deployments and uses them to update the deployment models of the IoT hub. After that, it utilizes the tagging mechanisms provided by Azure IoT Hub to control concretely which edge device should be deployed with which deployment model. The candidate deployments in the deployment pool are generated by the design-time diversifier, as presented in Section 3. The software artefacts used by these deployments are in a public artefact registry, such as Docker Hub, so that the devices are able to install these artefacts by themselves.

4.2.3.3 Detailed design

Main features

The first prototype of DivEnact showcases the following features. It is worth noting that even though all these features require low-level support from Azure IoT Hub, none of them is directly provided by Azure.

1. Appoint a universal deployment for all devices. All devices will be upgraded automatically when this universal deployment is updated. When a new device is bootstrapped, or an existing device is reconnected, they will be updated to this universal deployment. This is the baseline feature of the DivEnact. The only diversity comes from the fact that we allow the devices to be upgraded in an asynchronous way: the ones that are not currently online still use the old version, until they go online again. We implement this feature by creating a production deployment model, setting its target condition to be “all the production devices”, and tagging[15] all the devices as “production”.
2. Appoint a subset of devices for preview. These devices will be deployed with the “next version” of the application, so that the vendor can collect the user feedbacks before pushing it to all other users. The operator can select trial devices manually, or provide a certain condition to let the tool automatically select a specific sub-set of devices. This feature is implemented by creating a deployment model with the preview versions of the application modules, and tagging the appointed or selected devices with the “preview” tag. The operator can also opt to keep an appointed number of preview devices, which means that if a preview device is offline, a next available one will be switched to the preview mode.
3. Promote preview deployment to production. All the devices will be upgraded to the preview deployment. This is implemented by updating the condition of the preview deployment so that it applies to all the production devices, and tagging the preview devices back to “production”.
4. Diversify the applications based on a hardware condition. DivEnact maintains a set of equivalent candidate deployments, each of which applies to a specific setup of the edge device, including the CPU architecture of the edge device, the available resources, and the IoT devices connected to it. We implement this feature by adding another dimension of tags to the edge devices, and control the conditions of the deployments to include these tags.
5. Diversify the applications among identical devices. The design time diversifier can generate multiple versions of modules and architectures that are functionally equivalent. For the sake of security, operators can use DivEnact to shuffle these equivalent applications among the identical edge devices. We achieve this feature by generating an additional dimension of tags for the devices, and control the conditions of the deployment models, accordingly.

[15] Strictly speaking, the tag is added to the “digital twin” of the device, which is a model maintained by the IoT hub, and therefore the tagging operation does not require the device to be online.

6. Recover failed device. If a device has software failures, the user can reset the device to the factory setting, and once it again gets connected to the internet, the deployment manager will automatically deploy the application according to the newest relevant deployment model.
7. Rollback devices. If a deployment has problem on a device, an operator can remotely roll it back to the last working deployment, which is recorded as a specific deployment for this device.

Diversity management based on runtime models

We use a model-based way to manage the fleet of IoT sub systems. The DivEnact tool maintains a model at runtime that reflects the entire fleet, and the management is implemented as a set of reading and modification operations on this model. A reading operation of this runtime model will obtain the real time status of the fleet, i.e., devices and their deployments. On the other hand, a modification operation on the model will be directly reflected into changes to the managed subsystems.

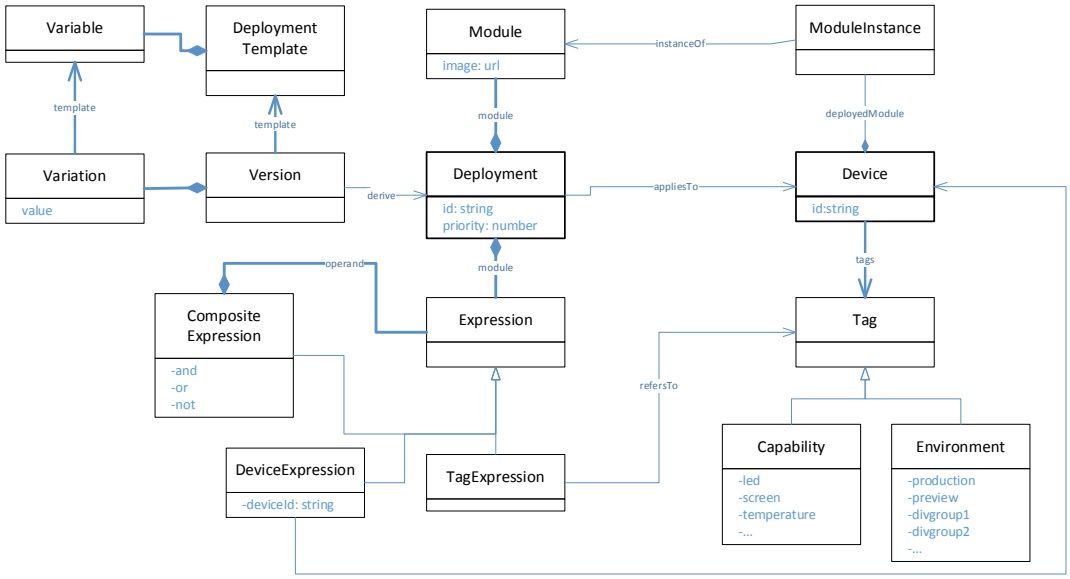

Figure 25. Meta-model for diversity control at runtime

The Figure 25 shows the meta-model of the fleet management model used by DivEnact. The central concepts of this model are Device and Deployment, representing an edge device and a deployment plan. A device may be deployed with several module instances, and currently we support module instances only in the form of Docker containers. A Deployment is a specification that composes a set of module types, and in our set up, each module type corresponds to a Docker image.

One fundamental design principle behind DivEnact is that we never require, or allow, developers to directly manipulate what module instances are deployed on an edge device. Instead, they can only specify a deployment plan, and if the plan matches a device, the IoT hub will automatically deploy the module instances according to the plan. This assumption fits well the requirements of an industrial situation such as the TellU use case, when their edge devices are scattered across different networks, are unstable in connection, and are therefore not guaranteed to provide a direct access from the administrator to the device. In order to highlight what operations are allowed to the DivEnact users, we highlight the relation that are writable by DivEnact as bold connections in the meta-model, and all the other connections are derived from these relations, and are maintained automatically by DivEnact at runtime. For example, following the design principle of no direct device operation, we allow the DivEnact users to decide what module types are included in a deployment plan (the module relation between Deployment and Module), whereas what module instances are running on the device (the deployedModule relation between Device and ModuleInstance) is automatically controlled by DivEnact, and not editable by the users.

The relation between deployment plans and devices are many-to-many: a deployment plan may apply to multiple devices (in some extreme cases, there could be only a single deployment plan that applies to all the devices), and multiple deployment plans may apply to the same device. In the latter case, the deployment with higher priority will be deployed to this device. DivEnact uses a tagging system to specify the mapping between deployment and devices, based on the condition-tag mechanisms provided by Azure IoT Hub. In particular, each device may have several tags, describing its nature with respect to different dimensions. We currently support two types of tags. Capability tags indicate the hardware features of a device, such as connectivity (Wi-Fi, Bluetooth, cellular connection), output devices (LCD screen, speakers), etc. Environment tags describe the application context and define whether a device is used for production, preview, testing, etc. Each tag type is an enumeration of user-defined options. A deployment plan contains a condition, which is an expression composing tag predicates and id predicates. The former refers to the tags given to the devices, while the latter directly refer to a specific device. If the condition evaluates to true, the deployment plan applies to the device, and DivEnact will automatically deploy the module instances into the device according to this plan.

In a fleet of IoT sub systems, it is typical that many deployments share a common structure, but use different version of some modules. Instead of asking users to specify each deployment from scratch, we maintain a set of deployment templates, each of which contains some variables with un-resolved values. Users can use a specific value for each variable to derive a deployment variation from the template.

We use the following sample scenarios to explain how this model-based diversity management approach works.

For the sake of simplicity, the application we deploy to the devices contains only one module, a container that lights up a simple light-emitting diode (LED) or a Sens Hat LED matrix wired to Raspberry Pi. The module has different versions, which indicate red, green and blue LED colors, respectively. We use this simplified application for the demo purpose, since it is easier to visually observe the effect of deployment changes.

The demo employs two deployment templates, one with a Docker Image that lights the LED, and another uses the LED matrix. The variable for both templates is the color. From the two templates, we define 6 variations: 3 for the LED, configured as red, green and blue, respectively; the other 3 for the LED matrix, also configured by same three colors. The demo has 5 devices - a sufficient number for a hands-on demonstration. Four devices are equipped with LEDs, the other one is equipped with an LED matrix.

At the first step, we set "led-red" and "matrix-red" as the production deployments, using the following command. A brief explanation of the command line user interface for DivEnact can be found in 4.2.3.5.

```
node build/main.js global production led-red, matrix-red
```

DivEnact will create two deployments:

```
(deployment1) production_led-red:
  condition: tags.environment=production and tags.capability=led
(deployment2) production_matrix-red:
  condition: tags.environment=production and tags.capability=matrix
```

After that, it will tag all the devices with environment: production. Within 3 minutes, we will see all the LEDs on the devices indicating red.

After that, we pick randomly 2 devices to preview an unstable version: led-blue

```
node build/main.js global preview led-blue
```

This simulates the situation of A/B testing for IoT systems: the vendor chooses a subset of customers to trial a new version.

After the led-blue version is confirmed, we can set it to production.

```
node build/main.js global production led-blue
```

Finally, if we are confident that all the three versions of the LEDs work fine and are functionally equivalent, we can shuffle them among the devices, in order to increase the diversity, and thus the security, of the entire IoT fleet.

```
node build/main.js global shuffle led-blue, led-red, led-green
```

This will lead to a more colourful fleet.

4.2.3.4 Interface

DivEnact interacts with other ENACT tools via the IoT Hub. Any tools, either deployed on the edge device or running as a backend service can send messages to the IoT Hub or subscribe the messages through the Hub, and in this way to communicate with the diversifier.

We are in the process of implementing the following connections with the other ENACT tools.

1. **With GeneSIS**. The deployment manager only supports the deployment of docker containers on a small scope of edge devices, i.e., x86-64 or ARM64 CPU with Windows or Linux OS. When we need to deploy an artefact on lower-end IoT devices, i.e., the "last-mile" deployment, we need to delegate this task to a deployment engine running on the edge device. For this purpose, we choose GeneSIS, running inside a docker container. In particular, DivEnact will deploy GeneSIS deployment engine into an edge device, and trigger the deployment operation via the so-called cloud-to-device message. In this way, we use GeneSIS to manage each sub-system that comprises a central edge device and a number of IoT devices, locally connected to it, and use the DivEnact to manage a resulting fleet of such sub systems, when these systems are not connected directly with each other.
2. **With security monitoring**. **DivEnact** uses the security monitoring tools to collect the security-related status of edge devices, and use these collected statuses to design the deployment and diversification plan. To achieve this, DivEnact requires the security monitoring tool to send the monitoring messages to the IoT Hub.

4.2.3.5 Tutorial

As the first step, DivEnact provides a command-line user interface. Developers interact with DivEnact via a set of commands through the terminal. We design the commands following a hierarchical style, inspired by mainstream DevOps tools, such as Docker and Git. At the same time, we are working on a web-based GUI using Ant.Design.

The use of DivEnact requires offsetting up an Azure IoT Hub cloud service, and a bootstrapping of all the managed Edge devices.

Create cloud resources
Follow this tutorial[16] to create an Azure IoT hub and add one (or more) IoT Edge Devices. Skip the steps for creating virtual machine (we will use real device) and deploy modules (this is what this tool will do, in a programmatic way).

Take a note of the following credentials:

- IoT hub connection string
- The edge device's connection string

[16] https://docs.microsoft.com/en-us/azure/iot-edge/quickstart-linux

Bootstrap edge device

1. SSH into the edge device
2. curl -L https://raw.githubusercontent.com/SINTEF-9012/divenact/master/edge/bootstrap/setup.sh -o setup.sh
3. echo '<Your device connection string>'>connection.credential
4. sudo bash ./setup.sh

Execute diversity management services

1. Go to the [service] folder
2. Copy the hub connection string into the azureiot.credential file
3. npm install
4. tsc
5. node ./build/main.js

Command-line interface

```
node ./build/main.js <command> [--version] [--help]
```

```
Top level Commands:
  device         Handle edge devices
  deployment     Handle deployments
  global         Handle devices and deployments together
  help [cmd]     display help for [cmd]
```

```
node .\build\main.js global <command>
```

```
Global Commands:
  production <variation>: set variant for production, and tag all devices into
'production'
  preview [options] <variation>: set variant for preview, and tag all or random
devices into 'preview'
    Options:
        -r, --random <N>  Preview on N random devices
  shuffle [variantions...]: randomly deploy the designated variantions into all
the devices
  query <queryString>: query device, deployment or module information using the
query string
```

```
node ./build/main.js device <command>
```

```
Device Commands:
  list|ls [options] : List all edge devices
    Options:
        -d, --details        show details
        -t, --tag <tagName>  show devices and their values of this tag
        -h, --help           output usage information
  tag [options] <id> [otherIds...]: Add/update tags to devices (appointed by id's,
or randomly with a number)
    Options:
        -e, --environment <value>  set environment tag
        -c, --capability <value>   set capability tag
        -r, --random               tag randomly N devices
        -h, --help                 output usage information
```

```
node ./build/main.js deployment <command>
```

```
Deployment Commands:
  list|ls [options]
    Options:
        -c, --conditions  show target conditions
        -h, --help        output usage information
  add [options] <variation>: add the deployment variation into IoT Hub
    Options:
        -e, --environment <value>  set the environment tag
        -h, --help                 output usage information
```

4.2.4 Other control mechanisms

This section describes the additional control services offered by ENACT Security and Privacy Monitoring & Control Enabler conceived as mechanisms that could be easily integrated in the IoT application components and activated at runtime when needed. These control mechanisms respond to the need of having security policies controlled as much as possible and to secure the access to resources and services in the IoT system components.

To this end, a PDP control agent has been developed that will be deployed by the GeneSIS Orchestration tool together with the components of the IoT system. This PDP control agent will be external to the SIS components and managed by the Control Manager in Figure 5. The agent is able to send events at the IoT application level to the streaming bus for their processing and correlation with other data from system and network layers for anomaly detection.

The PDP agent is a Policy Decision Point that evaluates pre-defined security policy rules over access requests where the attributes of the request context (e.g. requesting IP, user role, etc.) are no longer taken from the HTTP protocol as it is not used in the IoT systems of the ENACT use cases, but rather these attributes need to be passed to the agent together with the rule evaluation order.

The agent relies on XACML policy specification standard by OASIS and checks whether the rules evaluate to true or false. The enforcement of the access (grant or deny the access) will be done by an external entity (e.g. the IoT platform) according to the result. Note that the power of the access control performed depends on the granularity of attributes taken into account in the XACML rules. The finer the granularity the richer the possibility.

As a simple example, the following Figure 26 shows an extract of an access control policy based on XACML rules that evaluates the value of role attribute.

```
{
...
"apply": "deny-overrides",
"policies": [
{
    "rules": [
       {
       "target": [
              { "credentials:app_metadata.roles": "developer" },
              { "credentials:app_metadata.roles": "business_manager" },
              { "credentials:app_metadata.roles": "service_administrator" }
```

```
            ],
            "effect": "deny"
      }
    ],
    "apply": "deny-overrides",
    "target": [
    { "context:path": { "regexp": "/deployments/*"}  }
    ]
},
….
```

Figure 26. Example of XACML rules for access control policy

The interface of the PDP Agent being developed in ENACT will offer at least the following services:

- updatePolicy(policyID): updates the indicated XACML policy stored in the agent.
- evaluatePolicy(policyID, attributes): boolean evaluates the rules in the XACML policy for a specific set of attributes sent as parameters.
- start: starts the agent running.
- stop: stops the agent.

5 Conclusions

The goal of WP4 in ENACT is to provide support to define and ensure the secure, resilient and privacy-aware behaviour of smart IoT systems. The work package is dealing with support to development and operations phases of the DevOps cycle. At development phase the support mainly includes system security, privacy and resilience (diversity) requirements specification mechanisms together with the associated controls specification. At operations phase, the support is focused on continuous monitoring of possible security and privacy incidents and attacks to the IoT system as well as early reaction to them.

The main focus of this document, linked to the delivery of the prototypes of the tools provided through the work package enablers, is the current use case requirement coverage and the presentation of the operation of the different tools and how they can be used in the ENACT framework.

WP4 is developing the detailed design and implementation of the following enablers: i) a Security and Privacy Monitoring and Control enabler, which includes an advanced context-aware access control mechanism and ii) a Diversifier to be able to analyse diversity requirements of SIS and request software variants deployments when needed.

As described in the Table 2, some requirements related to security, privacy and resilience identified by the use cases are not yet completely fulfilled by the current prototype tools. The objective in the next step of the project is to improve this degree of fulfilment.

At this step, the various tools of the work package are not yet fully integrated all together. This will be addressed in the next step of the project.

The large-scalability aspect of IoT systems is a characteristic considered from the design phase of the WP4 Enablers, this capacity will be evaluated by series of tests emulating these environments.

6 References

6.1 Bibliography

[1] Rios, E., Iturbe, E., & Palacios, M. C. (2017, August). 'Self-healing Multi-Cloud Application Modelling'. Proc. Int. Conf. on Availability, Reliability and Security (p. 93). ACM.

[2] MUSA EU H2020 project, Deliverable D2.3, https://www.musa-project.eu/sites/musa3.drupal.pulsartecnalia.com/files/documents/MUSA%20D2.3%20Final%20SbD%20methods%20for%20multi-cloud%20applications.pdf.

[3] National Institute of Standards and Technology (NIST), 'Security and Privacy Controls for Information Systems and Organizations'. NIST SP-800-53, revision 4.

[4] ISO/IEC 27002:2013 — Information technology — Security techniques — Code of practice for information security controls (second edition)

[5] ISO/IEC 27017:2015 Information technology -- Security techniques -- Code of practice for information security controls based on ISO/IEC 27002 for cloud services

[6] ISO/IEC 27018:2014 — Information technology — Security techniques — Code of practice for protection of Personally Identifiable Information (PII) in public clouds acting as PII processors

[7] Cloud Control Matrix (CCM) Alliance, C.S.: Cloud security alliance, cloud controls matrix v3.0.1 (11-12-18 Update), https://cloudsecurityalliance.org/group/cloud-controls-matrix/

[8] Cloud Security Alliance: 'Consensus Assessments Initiative Questionnaire v3.0.1 (9-1-17 Update)', https://cloudsecurityalliance.org/download/consensus-assessments-initiative-questionnaire-v3-0-1/

[9] The OWASP Application Security Verification Standard (ASVS) Project, https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project

[10] Database Hardening Best Practices, Berkeley Information Security Office, https://security.berkeley.edu/resources/best-practices-how-articles/database-hardening-best-practices

[11] National Institute of Standards and Technology (NIST), 'Security and Privacy Controls for Information Systems and Organizations'. NIST SP-800-53, revision 5 Draft.

[12] ISO/IEC PRF 27552 Security techniques -- Extension to ISO/IEC 27001 and ISO/IEC 27002 for privacy information management -- Requirements and guidelines [Under development]

[13] Brice Morin, Jakob Høgenes, Hui Song, Nicolas Harrand, and Benoit Baudry. 2018. Engineering Software Diversity: a Model-Based Approach to Systematically Diversify Communications. In Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS '18). ACM, New York, NY, USA, 155-165. DOI: https://doi.org/10.1145/3239372.3239393

[14] Industrial Internet Consortium. Industrial Internet of Things Volume G4: Security Framework. IIC:PUB:G4:V1.0:PB:20160919.

[15] Noguero, A., Rego, A., & Schuster, S. Towards a Smart Applications Development Framework. Social Media and Publicity, 27.

6.2 Software tool repositories

[16] Open Source IDS / IPS / NSM engine Suricata, rules format documentation. https://suricata.readthedocs.io/en/suricata-4.1.4/rules/intro.html

[17] Elastic stack products, https://www.elastic.co/products/

[18] T-shark, Core network protocol analyzer of Wireshark, https://www.wireshark.org/docs/man-pages/tshark.html

[19] Elastic Filebeat Software, https://www.elastic.co/en/products/beats/filebeat

[20] Elastic Auditbeat, https://www.elastic.co/guide/en/beats/auditbeat/current/auditbeat-overview.html

[21] Open Source IDS / IPS / NSM engine Suricata, https://suricata-ids.org/

[22] Open Source IDS / IPS / NSM engine Suricata ruleset update management documentation, https://suricata.readthedocs.io/en/suricata-4.1.4/rule-management/suricata-update.html

[23] Apache kafka, https://kafka.apache.org/

[24] Elastic Logstash Software, https://www.elastic.co/en/products/logstash

[25] Elasticsearch Software, https://www.elastic.co/en/products/elasticsearch

[26] Elastic Kibana Software, https://www.elastic.co/en/products/kibana

[27] Apache Kafka clients, https://cwiki.apache.org/confluence/display/KAFKA/Clients

[28] About Docker CE installation documentation, https://docs.docker.com/install/