Melodic
Big data cloud

Multi-cloud Execution-ware for Large-scale Optimized Data-Intensive Computing

H2020-ICT-2016-2017
Leadership in Enabling and Industrial Technologies; Information and Communication Technologies

Grant Agreement No.:
731664

Duration:
1 December 2016 -
30 November 2019

www.melodic.cloud

Deliverable reference:
D5.03

Date:
30 June 2018

Responsible partner:
7bulls

Editor(s):
Paweł Skrzypek

Author(s)
Paweł Skrzypek, Yiannis Verginadis, Ioannis Patiniotakis, Christos Chalaris

Approved by:
Keith Jeffery

ISBN number:
N/A

Document URL:
http://www.melodic.cloud/deliverables/D5.03 Security Requirements & Design

Title:

Security requirements & design

Executive summary

Although cloud computing is a fascinating path leading to multiple facilities and tangible benefits for companies, it, at the same time, is very difficult to manage from a security point of view. The list of the main risks associated with clouds is very long, and even longer for multi-clouds; the assumptions of the Melodic project are focused on the following – the most important from the project's point of view – risks: improper identity management, credentials and access, and unsecured interfaces and APIs. Of course, the number of ways to deal with these problems is no shorter. Every customer that plans to implement cloud solutions needs to remember to use simultaneously a proper set of tools which addresses those risks, e.g.: data encryption tools, management of processing operations, identity and access, virtual firewalls and other virtualisation management tools, data loss prevention etc.

While designing Melodic, all above mentioned issues have been taken into consideration thus our project, from the beginning, addresses possible security problems that may occur during usage. We had to be aware that the security risks may be completely different for various sectors, like industry, public administration or financial. Moreover, there is a need to answer different type of questions from companies representatives, for a Financial Director might have other priorities than a Technical Director.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Document	
Period Covered	M12-18
Deliverable No.	D5.03
Deliverable Title	D5.03 Security requirements & design
Editor(s)	Paweł Skrzypek
Author(s)	Ioannis Patiniotakis, Paweł Skrzypek, Yiannis Verginadis, Christos Chalaris
Reviewer(s)	Jörg Domaschka
Work Package No.	5
Work Package Title	Integration and security
Lead Beneficiary	7bulls
Distribution	PU
Version	Final
Draft/Final	Final
Total No. of Pages	41 + Appendices

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table of Content

1 Introduction 6
2 Structure of the document 7
3 Security requirements for Melodic platform 7
3.1 Cloud providers' credentials security 8
3.2 User and component authentication 8
3.3 Access Control 8
4 Conceptual overview of security in the Melodic architecture 9
5 Melodic Authorization Service 10
5.1 Melodic Authorization Service Requirements 10
5.1.1 "Access Control" to Melodic platform components use-case 11
5.1.2 Pre-authorisation of Application deployment & Data placement (plans) use-case 12
5.1.3 Requirements for Authorisation Service 13
5.2 Related work on Access Control 17
5.3 Authorisation Service Design 18
5.3.1 Attribute-Based Access Control (ABAC) model 21
5.3.2 XACML model 22
5.3.3 Use of Aspect and Aspect-Oriented Programming 25
5.3.4 Use of Request Interceptor for Spring-boot based components 27
5.4 Authorisation Service Architecture 28
5.4.1 Attributes in Authorisation Service 29
5.4.2 Authorisation Service Architecture 29
6 User Authentication Service 33
6.1 User Authentication Service Requirements 33
6.2 Related work on user authentication 34
6.2.1 Monolithic applications 34
6.2.2 Distributed Session Management 35
6.2.3 Token-Based Authentication 35
6.3 User Authentication Service Design Decisions 36
6.3.1 User credentials store 36
6.3.2 Authentication mechanism – token-based 36

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

6.3.3 Changes in process flow 37
6.3.4 Changes in components' methods invocation 37
6.4 User Authentication Service Architecture 37
7 Cloud providers' credentials security 38
7.1 Cloud providers' credentials security requirements 38
7.2 Cloud providers' credentials security design decisions 39
8 Summary 40
9 References 41
Appendix A: Assessing the Melodic Security Services via External Security Experts ... 42
A.1 Summary of the external experts' assessment 43
A.1.1 Positive highlights 43
A.1.2 Recommendations for enhancement 44
A.2 Next steps for the Melodic security services enhancement 48
Appendix B: Security Audit Report by Prof. Antonis Michalas 52
Appendix C: Security Audit Report by SIDIO Sp. z o.o. 73

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Index of Figures

Figure 1: Access Control use case diagram 12
Figure 2: Deployment plan Pre-authorisation use case diagram 13
Figure 3: XACML Flow & Architectural Components 23
Figure 4: Aspect-Oriented Programming 26
Figure 5: Tomcat request processing cycle 28
Figure 6: Attribute flow in Melodic Authorisation service 29
Figure 7: Authorisation Service architecture 30
Figure 8: Authorisation service within Melodic platform architecture 32
Figure 9: The authentication flow of operations 36
Figure 10: Cloud Providers credentials flow 40

Index of Tables

Table 1: Use case to Authorisation Requirements mapping 17
Table 2: Use case to Authorisation Requirements mapping 18
Table 3: Java-based, open source, XACML tools 24
Table 4: User authentication security requirements 34
Table 5: Cloud providers' credentials security requirements 39
Table 6: Consolidated details of the recommendations provided by Prof. Michalas 44
Table 7: Consolidated details of the recommendations provided by the SIDIO experts . 46
Table 8: Related recommendations 48
Table 9: Addressing the experts' recommendations 48

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

1 Introduction

Security is one of the most crucial elements of cloud solutions. Based on the cloud adoption survey[1], it is one of the most important obstacles towards migration to the cloud. That is the reason why several tasks in the Melodic project are dedicated to the topic. The results of the work performed over these tasks are reported in this deliverable.

There are many security topics related to the Melodic project: The first one concerns the security of user-provided cloud provider credentials (access and storage). Based on the underlying frameworks (PaaSage), the security of these credentials has been increased. In particular, cloud providers' credentials are now stored in a secure way. Further, they are stored in only one component of the platform (Cloudiator), i.e., in the exact place where they will be also used.

The second element related to security concerns user and component/external system authentication. To achieve this, a state-of-the-art solution based on SAML2[2] and LDAP[3] has been implemented. The authentication is based on generated tokens, valid for a certain period of time, instead of communicating the user/password credentials in a non-secure and textual manner in each method invocation.

Finally, an advanced authorisation module using the XACML[4] standard and WSO 2 Balan[5] authorisation platform has been designed, implemented and integrated in the Melodic platform. Thanks to that, a very high level of security control has been achieved, with the ability to configure a very flexible and complex set of security rules for authorisation of the selected operations.

In this way, the described security enhancement of the Melodic platform leads to a significantly better security level with respect to the one exhibited by the underlying frameworks that are utilised, like PaaSage.

[1] https://www.forbes.com/sites/louiscolumbus/2017/04/23/2017-state-of-cloud-adoption-and-security/#657fdbe21848
[2] https://access.redhat.com/documentation/en-US/Red_Hat_JBoss_Portal/6.1/html/Administration_and_Configuration_Guide/chap-Security_Assertion_Markup_Language_SAML2.html
[3] https://ldap.com/
[4] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
[5] http://xacmlinfo.org/category/balana/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

2 Structure of the document

The structure of this document is as follows:

Chapter 3: Security requirements for Melodic platform – general overview of security requirements for the Melodic platform.

Chapter 4: Conceptual overview of security in the Melodic architecture – general overview of security related elements in the Melodic architecture.

Chapter 5: Melodic Authorization Service – Requirements, architecture and design decisions related to the *Authorisation Service* in Melodic.

Chapter 6: User Authentication Service – Requirements for the User Authentication Service as an important component of the Melodic platform that allows, together with the authorisation service, accessing and operating the Melodic platform system.

Chapter 7: Cloud providers' credentials security – Requirements and design decisions related to cloud provider's credentials security in Melodic.

Chapter 8: Summary – summary of the document with conclusions and future work directions.

Appendix A: Assessing the Melodic Securit Services via External Security Experts – an introduction to and a summary of an assessment of the security mechanisms of the Melodic platform conducted by external security experts.

Appendix B: Security Audit Report by Prof. Antonis Michalas

Appendix C: Security Audit Report by SIDIO Sp. z o. o.

In the document there are references to the Melodic architecture which is described in detail in the D2.2 deliverable "Architecture and initial feature definition" [1].

The target audience of this deliverable are technical partners involved in development of the Melodic platform. Also, the requirements and security capability of the platform would be beneficial for use case partners and all users of Melodic.

3 Security requirements for Melodic platform

This section presents key security requirements for the Melodic platform, based on the experience from the PaaSage project, the Melodic project's Description of Action, requirements from use cases applications and the general experience of developing and maintenance of Cloud Computing based applications and IT systems.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

3.1 Cloud providers' credentials security

Based on experiences from underlying frameworks (mostly the PaaSage framework), the security of cloud providers' credentials needed to be improved. The credentials should not be stored in plain text, but in an encrypted form. The credentials should be stored in an encrypted form, using a symmetrical method of encryption with a secret key. The secret key used to encrypt a password should be known only to platform administrators. Also, the credentials should be stored in just one place in the system, the place where they are really needed to be used. Users should provide his/her credentials only once, in the initial stage of application deployment before the deployment process starts.

3.2 User and component authentication

The Melodic platform should use a unified method for user and component authentication at the platform/system level. Each operation originated by a user, a component or an external system should be properly authenticated, using the proven method for that purpose. It should also be as secure as possible. In particular, authentication should be based on username and password, but user and component credentials should be stored in one place, not spread across many components. Authentication per method invocation should be based on tokens, using industry proven security standards like SAML2. Using token-based authentication does not require to pass the user credentials each time, for each operation invocation and improves the security level.

3.3 Access Control

The Melodic platform should also use a unified method for user and component authorisation at the platform level. Authorisation ensures that only eligible entities (users or components) can access protected platform resources and apply certain operations on them. Each access attempt to a resource is checked against a set of access control policies, captured using the XACML language, which is the de facto standard. During authorisation checking, various stated and contextual information should be used; this information relates to the requestor (user or component), the resource being accessed (data, methods etc.), the attempted operation as well as other environment data (date/time, Upperware operational status, etc.).

Beyond access control, the authorisation infrastructure should be consulted on whether a given application deployment plan, generated by the Upperware, complies to a set of deployment policies. Such policies may encompass constraints and limitations referring to application deployment (for instance total cost or number of virtual machines deployed).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

4 Conceptual overview of security in the Melodic architecture

Melodic, as its significant part is an integration of the underlying frameworks, needs to address security concerns which have been raised based on exploitation of these frameworks. Also, the introduction of new components requires additional security enhancements.

The most important shortcomings related to the security of underlying frameworks are listed below:

1. Cloud providers' credentials passed and stored in plain text, without encryption;
2. Lack of centralized user authentication;
3. Missing centralized inter-component authentication;
4. Lack of possibility for creation of advanced authorization rules.

The work related to security in Melodic has been conducted in the three directions listed further in this chapter. All these directions are based on the requirements described in chapter 3.

There are three main security elements covered in Melodic:

1. Cloud providers' credentials security – handled by the BPM process and Cloudiator. This element of security is analysed in chapter 7.
2. User authentication service – handled by the JWT and SAML2 tokens component. It is used for each method invocation. This component of security is detailed in chapter 6.
3. Access control authorization – handled by XACML and Balana server. This element of security is analysed in chapter 5.

Each above-mentioned security element is presented in a different section of the deliverable. Due to the different nature and specificities of these components, the content and structure of these sections are slightly different.

In the Melodic platform the above security elements have been implemented and harmonized together, to address security shortcomings from previous projects and achieve as good results in terms of security as possible.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

5 Melodic Authorization Service

This chapter documents the Melodic's *Authorisation Service*. Authorisation refers to a security mechanism that determines and enforces access privileges of a requesting entity, related to resources and application features. In Melodic, this service materialises two objectives. First, the supply of a security-by-design access control framework for Melodic platform components, which enables the adequate protection of sensitive platform resources (such as services, components, workflows, and data), both from unauthorised access attempts as well as from compromised or misbehaving platform parts.

The second objective relates to the enforcement of policies and limitations regarding the deployment of Melodic applications and their data in cloud providers. Normally, the Melodic reasoner will produce correct deployment plans conforming to any given constraints and limitations. However, a compromised, due to a cyber-attack, Upperware component could possibly yield invalid deployment plans. Therefore, a precautionary validation step, before the actual deployment, would reduce the likelihood of deploying an application in a non-conformant manner. We refer to this step as pre-authorisation. The enforced limitations can be regulatory, corporate, as well as budget-, resource- or security-related.

These two objectives are quite different in their business purpose and involve different authorisation rules. However, the same authorisation capabilities and toolset can be used in order to achieve both of them.

This chapter is structured as follows. In subsection 5.1 the requirements for the Melodic *Authorisation Service* are extracted on the basis of two use cases, related to the two aforementioned objectives. Subsection 5.2 gives a brief overview of the most frequently used access control models introduced in the literature. Based on the information from the first two subsections, the *Authorisation Service* design decisions are given in subsection 5.3. Eventually, subsection 5.4 presents and details the Authorisation Service's architecture and implementation.

5.1 Melodic Authorization Service Requirements

Authorisation in the context of the Melodic project is seen from two different perspectives, based on two use cases corresponding to the two objectives mentioned above; namely, the “access control” to various Melodic platform components, and the “pre-authorisation” of application deployment and data placement plans in cloud providers. In the former use case, authorisation capabilities are considered to be responsible for protecting the platform itself from illegal access attempts and

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 731664

interference with its normal operation. In the latter case, authorisation capabilities refer to the pre-authorisation of application deployment and data placement plans, produced by Melodic Upperware, considering given set of policies, constraints or limitations. Each use case is discussed in more detail to help identify the relevant requirements for the *Authorisation Service*.

5.1.1 "Access Control" to Melodic platform components use-case

The Melodic platform comprises a set of network-connected micro-services, distributed over an intranet or a (virtual) private network. Despite the significant advantages of this approach, certain attack vectors exploiting the networked and distributed nature of the platform are possible. In order to ensure a sufficient level of security, it is necessary to protect platform components (micro-services) from unauthorised access attempts (either from within the platform or the outside world) and isolate those that have been compromised. However, detecting a hacked component cannot solely rely on presenting valid credentials, since they might be leaked, or a hacker might take control of a component requesting access or even pretending to be a platform component. For this reason, additional parameters must be taken into account; for instance, the components' previous behaviour (recorded in logs), the origin and time of an access request, or the current state and environment of the platform. Such information is usually termed as *context*. Dey and Abowd [2] define *context* as "any information that can be used to characterise the situation of an entity". Contextual information can be of various types and originate from diverse sources. Moreover, it might vary among Melodic adopters and applications. The Melodic Metadata Schema (introduced in deliverable D2.4 [3]) provides a classification of these information types, in its Context-aware Security model. This classification acts as a common vocabulary (between the components) for collecting, storing and leveraging information for authorisation purposes.

Access request data (data included and explicitly stated in an access request) and contextual information need to be combined and correlated in order to conclude whether an access attempt is legal or not, for instance checking whether an access request is part of the regular workflow (of Upperware) or an out-of-band access attempt. Another check is whether access is attempted at the right time and sequence (i.e., after prerequisite steps have been taken). Checks might combine information including the requestor identity, location, privileges, the resource identity and state, the intended operation (on the resource), as well as the time frame or other contextual information. It is also noteworthy that authorisation rules might change over time to satisfy new needs or fix problems that have been identified. Subsequently, the information needed and the way it is combined need to change accordingly. Figure 1 provides the use case diagram of the access control use case. Access control involves the collection of request data, contextual information and the use of access control policies.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

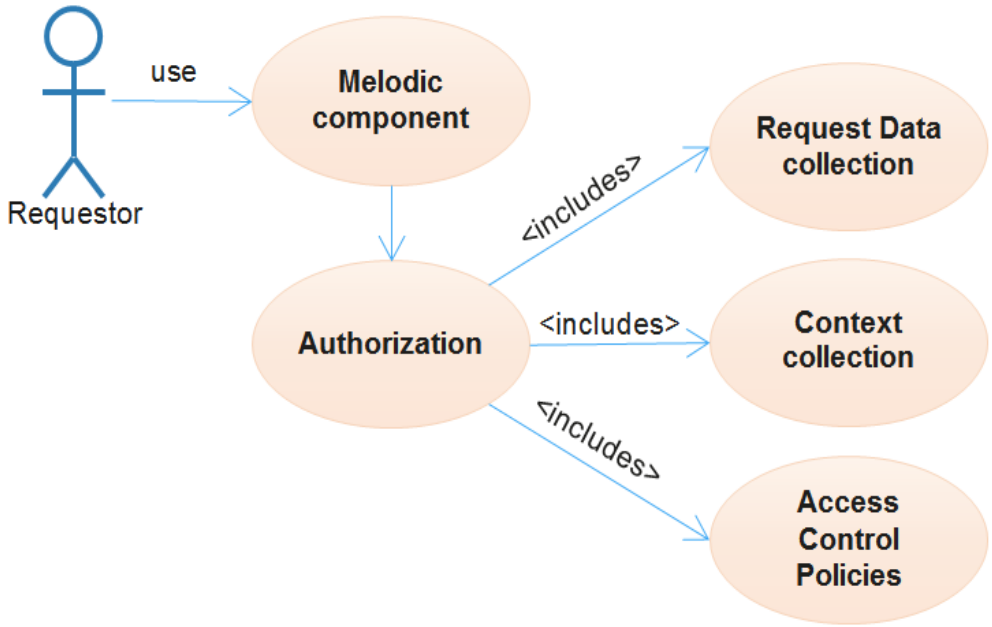


Figure 1: Access Control use case diagram

As already mentioned, several of the Melodic platform components need to be protected from unauthorised access. Since the platform is distributed (components might be installed in separate physical or virtual hosts), the authorisation capabilities must be present in every component that needs to be protected.

5.1.2 Pre-authorisation of Application deployment & Data placement (plans) use-case

Application deployment and data placement plans typically encompass information and instructions about the number and type of application components, the distribution of application dataset in VMs, the selected cloud providers, VM and data requirements (including security) and various setup procedures. Executing these plans leads to the deployment of operational multi-cloud applications. However, limitations and constraints might occur affecting the way an application must be deployed, operated and how data must be stored and processed. These limitations may vary between different geographical or logical regions and evolve. An example is the General Data Protection Regulation (GDPR)[6], which is put into effect in EU on the 25th of May 2018. Apart from regulatory and legal constraints, corporate standards and policies may also apply, as well as particular budget and resource constraints or rules of usage (e.g., the number of deployed VMs per cloud provider, cost of deployed VMs and stored data).

[6] https://www.eugdpr.org/the-regulation.html

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

As a consequence, a variety of information is required to ensure whether any given application deployment and data placement plans abide by a set of established policies (regulations, rules, constraints, and limitations). As in the "access control" use case, information needs to be combined to conclude whether a given plan satisfies the relevant policies. Furthermore, such information might change over time. The context classification and related concepts included in Melodic's Metadata Schema can also be used for application deployment plan pre-authorisation. Figure 2 provides the use case diagram of the pre-authorization use case. Pre-authorization pertains to Adapter and DLMS components of the Melodic platform. It involves the collection of request data, contextual information and the use of access control policies.

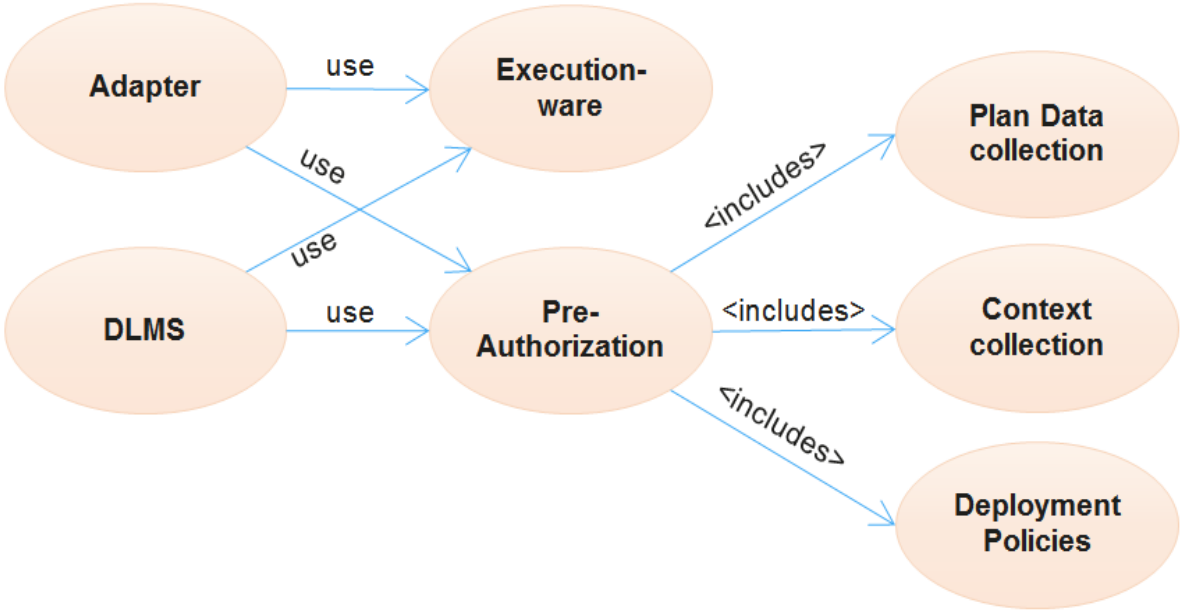


Figure 2: Deployment plan Pre-authorisation use case diagram

A pre-authorization policy could for example pose a limit on the number of virtual machines deployed on a cloud provider, or require the storage of data of a certain type to be stored in nodes located in EU.

5.1.3 Requirements for Authorisation Service

Melodic's authorisation service requirements have resulted mainly from three sources:

1. the two objectives presented above and the needs of the corresponding use cases,
2. the Melodic platform generalised requirements presented in deliverable D2.1 [4], especially those relating to privacy & confidentiality, and
3. the interaction with consortium partners (use case and technical partners).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

In order to satisfy the aforementioned objectives, generalised requirements and needs, the Melodic's authorisation service must exhibit specific capabilities and meet specific requirements, which are detailed in the following paragraphs. These requirements further elaborate on those specified in D2.1 [4] or are implied from the aforementioned use cases.

[R1] **Support for multiple information types and sources.** As already discussed, the identity, role, and credentials (user/password, clearance level or security labels) of the entity attempting to access a resource, are not adequate to determine whether such an access request is legal or not. The same holds for pre-authorising application deployment plans. In fact, a multitude of information is required, which either might be stated with the access request or derived or acquired from the *context* (for instance access time, requestor IP address or state of the resource). This information can be considered being in the form of attributes that characterise the entity attempting the access, the resource being accessed, the requested operation on the resource, the request object itself, or any other platform or environment entity that might affect the decision to allow or block the access attempt.

[R2] **Support for multiple authorisation check points.** Since the Melodic platform comprises several components, it is necessary to introduce access control checks at several points, where critical operations take place or sensitive data are stored or processed. Such points are the Upperware Control Plane, the Adapter and the DLMS components (which jointly implement the application and data deployment plans).

[R3] **Minimal changes to pre-existing platform code.** In order to enhance the usability of the authorisation service, it is essential to require minimal code changes when incorporating such security capabilities.

[R4] **User-defined and flexible authorisation.** The authorisation requirements may significantly vary between different Melodic platform installations, both regarding access control as well as regarding application deployment policies. It becomes apparent that the Melodic platform adopters must have tools at their disposal for capturing these requirements (as policies and rules). For this reason, a suitable language, capable of expressing complex relations between the various access request artifacts and their attributes, must be chosen. Additionally, a desirable (but not required) feature would be the ability to modify access control and pre-authorisation rules at runtime, without needing to restart the Melodic platform or its authorisation service.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

[R5] **Authorisation rules decoupled from code.** Access control and pre-authorisation rules may significantly vary. Furthermore, they might evolve over time. For this reason, they must not be hard-wired or tightly coupled with the platform code to avoid the need to update, compile and re-deploy the Melodic platform every time a rule changes.

[R6] **Leverage Melodic Metadata Schema.** In deliverable D2.4 [3], we have presented and detailed the Melodic's Metadata Schema, which serves as a common, versatile vocabulary for all Melodic software components and models. One of the constituting parts of the Metadata Schema is the Context-Aware Security model, which captures in an extensible manner the attributes of the notions involved in various access control and security scenarios. These notions are: The Subject (entity attempting access), Object (resource being accessed), Request (the access attempt artifact), Security Context Element (any attribute, either stated or contextual), Handler (attribute handling entities), Permission (rights to perform specific actions on resources) and Context Pattern (for recurring and complex access requests). In this respect, the *Authorisation Service* must take the Melodic's Metadata Schema under consideration.

[R7] **Availability and fault tolerance.** Authorisation capabilities must always be available for monitoring access attempts continuously and authorising only the legitimate ones. They must also be able to cope with various types of errors (including network errors) without going out of service or taking wrong decisions. Especially, they should be able to rapidly recover from faults and unexpected crashes.

The aforementioned requirements satisfy and further elaborate the "privacy and security" requirements presented in deliverable "D2.1 System Specification" [4]. Namely:

i. **Secure and context-aware data access control mechanism.**
This is covered by requirement "[R1] Support for multiple information types and sources" since it requires taking into consideration any type of information, including contextual and environment information, during the authorisation process. Moreover, "[R6] Leverage Melodic Metadata Schema" relates to this D2.1 requirement since it mandates the use of the Context-aware Security Model to classify the information used for authorisation.

ii. **Ability to accept user-defined data security and confidentiality requirements**. Requirement "[R5] Authorisation Rules decoupled from code" requires the authorisation rules to be held separately from code (thus enabling their change without modifying software) while requirement "[R4] User-defined and flexible authorisation" specifies that authorisation rules can be user-defined and that a language should be used for capturing them.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

The generic requirements stated in D2.1 may also apply to the authorisation service.

i. **Stability**.
   The system should work in a stable manner.
   Service stability should be ensured using suitable test cases.
ii. **Error handling**.
   Functional and non-functional errors should be properly handled.
   All service errors must lead to authorisation failures to ensure that anyone accessing sensitive resources is always properly authorised to do so.
iii. **Monitoring & traceability.**
   It should be possible to monitor and track all activities in the system.
   Authorisation service should generate detailed audit trails to enable effective monitoring and tracing.
iv. **Ability to deploy applications based on a high availability/disaster recovery configuration.**
   Authorisation service must be able to always respond under high load and also work in disaster recovery (configuration) mode
v. **Logging.**
   Support for unified logging of all components with configurable logging levels. Services must follow the same logging approach and share the same logging configuration with other Melodic platform components.
vi. **Backup.**
   Support for backing up system databases and critical components. It must be possible to backup authorisation rules and configurations.

The non-functional requirements presented in the same deliverable, D2.1 [4], are also relevant. These requirements are Extensibility, Reusability, Documentation, Quality, Fault Tolerance and Scalability. Authorisation service must be written in a modular way and provide an extension framework, thus making it easily maintainable, extensible and thus reusable. Additionally, it should be possible to use it in isolation from the rest of the Melodic platform, hence strengthening its reusability. Documentation is also needed to allow users and developers extend and configure the service for their own purposes. Suitable tests are required to ensure its quality and stability during development, and also for verifying a deployment in a production environment. Scalability should also be considered in order to ensure the uninterruptible and performant operation of authorisation service even with increased workload.

Eventually, the mapping of the two use cases (see subsections 5.1.1 and 5.1.2) onto the authorisation service-specific requirements ([R1] to [R7]) is given. It is interesting to note that the "access control" use case subsumes the "deployment plan pre-authorisation" use case with regards to requirements, which is depicted in Table 1 below.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 1: Use case to Authorisation Requirements mapping

Requirement	Access Control use case	Deployment Plan Pre-Authorisation use case
[R1] Support for multiple information types and sources	✓	✓
[R2] Support for multiple authorisation check points	✓	✓
[R3] Minimal changes to pre-existing platform code	✓	
[R4] User-defined and flexible authorisation	✓	✓
[R5] Authorisation Rules decoupled from code	✓	✓
[R6] Leverage Melodic Metadata Schema	✓	✓
[R7] Availability and fault tolerance	✓	✓

5.2 Related work on Access Control

Several access control models have been proposed in the literature and used in software products. These models provide a framework and a method of how resources, requestors, operations, and rules may be combined to produce and enforce an access control decision. Each model has certain advantages and disadvantages. The most well-known from these models are the following:

- **Discretionary Access Control (DAC).** In discretionary access control, the owner of a resource specifies which entities can access the resource. Most operating systems and file systems are based on this model kind. [5]
- **Mandatory Access Control (MAC).** In mandatory access control, all entities are given a security clearance (for example top secret, secret, confidential, unclassified), while also resources are given a security classification (top secret, secret, confidential, unclassified). When a request to access a protected resource arrives, the system checks if the clearance level of the requesting entity matches or surpasses the classification level of the resource. [6]
- **Identity-Based Access Control (IBAC).** In identity-based access control, mechanisms, such as Access Control Lists (ACLs), are used to capture the identities of the entities having the permission to access specific resources. If a requestor presents a credential matching an identity held in the ACL, he/she is allowed to access the corresponding resource. Each resource needs its own ACL.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Privilege sets must be assigned to each entity that needs to access a resource [6]. A disadvantage of the IBAC model is the increased effort required to create and maintain the ACLs of resources. Failing to update privileges correctly may end up with entities being able to access resources that they should not access.

- **Role-Based Access Control (RBAC).** It employs pre-defined roles associated with specific privileges. Entities are then assigned to roles (e.g., the role of Manager) in order to have access to resources. Resources require specific privileges (or roles) in order to be accessed. When an access request is received, the access control mechanism checks if one of the roles assigned to the entity requesting access and the set of privileges that this role is carrying match the privileges required by the resource being accessed. The RBAC model provides an easier and centralised management of access control than IBAC and reduces the need for ACLs. [7]
- **Attribute-Based Access Control (ABAC).** It uses policies that comprise rules, which in turn comprise logical conditions on many attributes. Typically, each rule contains at least a condition (boolean expression) and a decision (permit or deny) to take when the condition is true. Policies combine the outcomes of rules and yield the final decision using certain outcome combination methods. Attributes can be properties of the requesting entity, of the resource being accessed, of the operation requested, or any other contextual information.

ABAC models require less effort to create and maintain than RBAC and ACLs do [7], since they aggregate all authorisation rules in one place, i.e., the policy. When an access request is made, an ABAC-compliant engine will make an access control decision based on the available attributes and a given set of policies. Policies can be created and managed without directly affecting entities and resources, while entities and resources can be provisioned without affecting policies.

5.3 Authorisation Service Design

Using the requirements presented in subsection 5.1.3 as guidelines, we will subsequently give the Authorisation Service design decisions and provide brief justifications for our choices in Table 2 below.

Table 2: Use case to Authorisation Requirements mapping

Requirement	Design Decision
[R1] Support for multiple information types & sources	The ABAC model has been selected mainly due to its generality and flexibility. Furthermore, it considers multiple information types in the form of attributes as well as multiple attribute sources (more information in subsection 5.3.1)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

[R2] Support for multiple authorisation check points	A client-server architecture will be used in the *Authorisation Service*. Authorisation clients will be introduced at every point in the Melodic platform requiring authorisation evaluation, contacting for this purpose an authorisation server. This is in line with the ABAC paradigm. It is expected that both clients and the server will be on the same intranet or private network. Therefore, communication will be fast and more secure.
[R3] Minimal changes to pre-existing platform code	Three approaches are considered. For each platform component requiring authorization, the most suitable from these three approaches should be selected. • Use of an aspect-oriented technique to "inject" the authorisation client at suitable annotated code points. This approach enables the development of secure-by-design software, since it allows "tagging" the sensitive operations with appropriate authorization annotations. It requires relatively small code updates and re-compilation. More information will be given in subsection 5.3.3. • Use of a web server interceptor to "intercept" incoming HTTP requests and handle them to the authorisation client before allowing their normal processing. This approach does not require code changes, but requires reconfiguring the Tomcat server and updating the software package appropriately. More information will be given in subsection 5.3.4. • Direct use of authorisation client objects. This approach requires significant code additions and re-compilation. It is suitable for particular usages (for instance, writing a custom authorisation client) or when the previous two approaches are not applicable. More details on the three approaches will be given in the subsequent subsections.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

[R4] User-defined and flexible authorisation	The use of the XACML[7] language enables the specification of user-defined authorisation policies, as well as their maintenance and evolution, even at runtime. Moreover, the ABAC model[8] (followed by XACML) enables the use of any kind of attributes, acquired both from access requests as well as the environment. This fact, combined with the capability of the XACML language to capture complex attribute expressions, gives great flexibility in defining advanced authorisation rules and policies.
[R5] Authorisation rules decoupled from code	No authorisation rules or policies will be part of Melodic's applications code. Instead, policies will be captured separately (e.g., in one or more files). This is in line with the ABAC model and the XACML reference implementation.
[R6] Leverage Melodic Metadata Schema	The Melodic's Metadata Schema must be used as background knowledge, capturing all authorisation related attributes.
[R7] Availability and fault tolerance	Authorisation server clustering and client-side load-balancing; Server clustering will allow operating several instances of the authorisation server thus ensuring its high availability and continuity of service. Load-balancing allows distributing the service workload in several authorisation server instances, thus achieving better performance and lower response time.

Apart from the design decisions imposed by the requirements, a few more decisions have been made regarding the (technical) structure of the service:

- Use of a modular software structure, with separation of concerns. It will allow easier development and maintenance of service parts.
- Use of a plugin framework for introducing extension points and enabling the addition of functionality that might vary between adopters (for example, custom request data and context collection, environment context collection, and context storage).

[7] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
[8] https://www.axiomatics.com/100-pure-xacml/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

- Use of configuration files for including, activating and setting up features, such as plugins to use, security certificates for encrypted communication, database and servers to communicate with, and load balancing.
- Use of a Spring-boot framework for the *Authorisation Service* implementation to be in line with the rest of the Melodic Upperware components.

5.3.1 Attribute-Based Access Control (ABAC) model

In this subsection, we will justify our choice to use the ABAC model for the *Authorisation Service* and specifically the XACML model and language. Moreover, we will select an XACML-compatible engine to use in the *Authorisation Service*.

ABAC is an "access control method where subject requests to perform operations on objects are granted or denied, based on assigned attributes of the subject, assigned attributes of the object, environment conditions, and a set of policies that are specified in terms of those attributes and conditions" [8]. For the sake of completeness and clarity, the definitions of the terms "attribute," "subject," "object," "operation," "policy" and "environment" are given below.

- **Attributes** are characteristics of the subject, object or environment. Attributes contain information given in the form of name-value pairs.
- **Subjects** are human users or system entities, such as a device or piece of software, which attempt to perform operations on objects. Subjects can have one or more attributes.
- **Objects** are controlled system resources, such as devices, files, records, tables, processes, programs, networks, or domains containing or receiving information or being invoked in order to provide a service. In this sense, an object can be anything upon which an operation may be requested and performed.
- **Operations** are executions of specific actions on objects at the request of a subject. Operations include read, write, edit, delete, copy, and execute.
- **Policies** are sets of rules that enable determining whether an access request should be allowed, based on the attribute values of the subject, the object, operation and possibly the environment conditions.
- **Environment** represents the operational or situational context in which access requests occur. Environment context are detectable environment characteristics modelled as attributes. Environment characteristics are independent of the subject or object and may include the current date/time, location of users or the current system state.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Any ABAC-compliant system must implement the following conceptual workflow:

- The subject performs an access request for a specific operation on a specific target object,
- An ABAC-compliant engine retrieves policies from the policy repository and obtains the attributes required,
- This ABAC-compliant engine retrieves attribute values from various sources (including the access request itself), pertaining to the subject, object, operation, and environment,
- The ABAC-compliant engine uses the attribute values to evaluate if the access request complies with relevant policies and makes a decision on whether to permit or deny the respective requested access.

5.3.2 XACML model

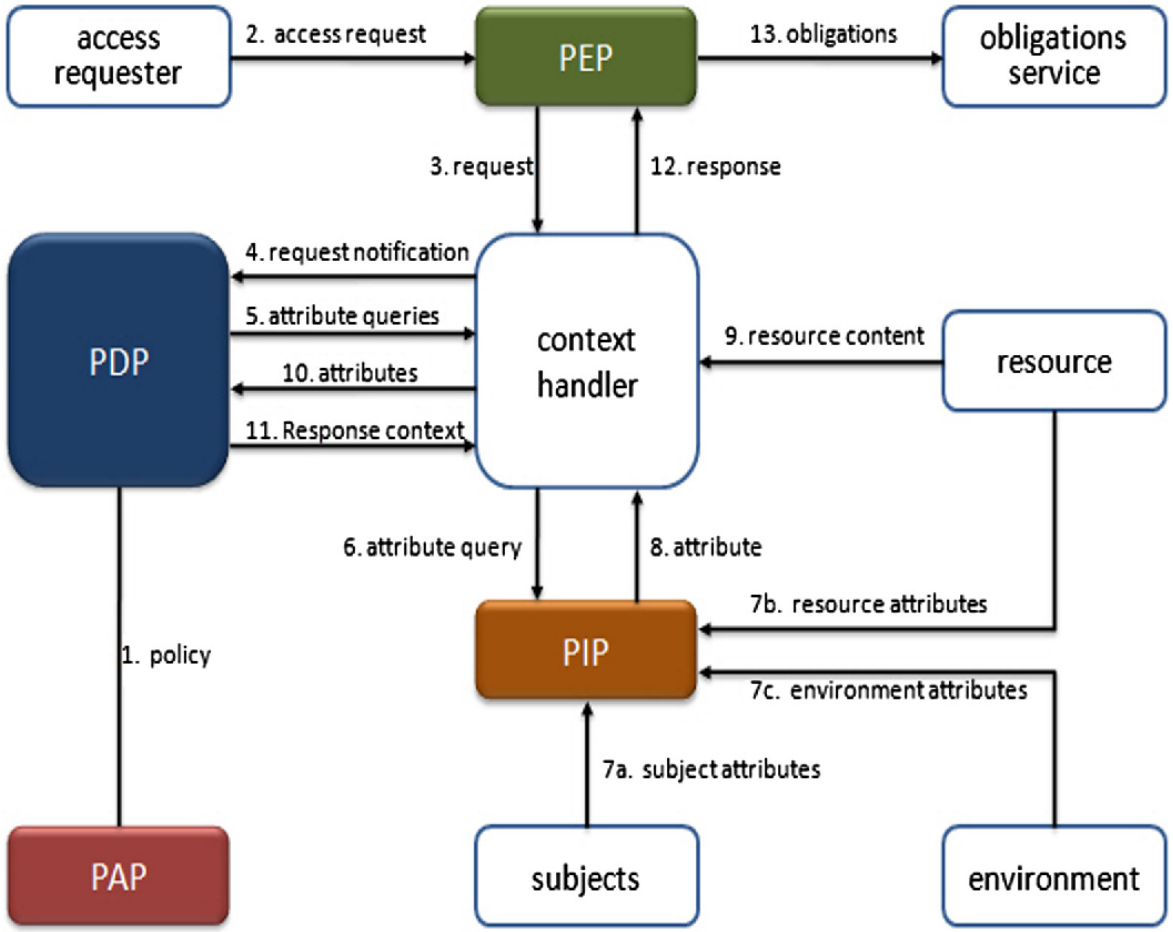
There are a few reference implementations of the ABAC model, but the most important one is the eXtensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC) [9]. XACML seems to be most widely used as it enjoys worldwide industrial adoption in sectors like banking, healthcare, and insurance. Moreover, most related products and vendors support it. As already stated, XACML has been selected for the Melodic *Authorisation Service*.

XACML is an XML-based, open-standard language promoted by OASIS, for expressing authorisation policies (as access control requirements) and querying access to resources[9]. Along with the language an access control architecture and a processing model is also proposed. Evaluating an access request to a resource, with regard to an XACML policy, may result in one of these four values: Permit, Deny, Indeterminate (an error occurred or needed values were missing) or Not Applicable (no related policy found).

The XACML specification defines five main components (Figure 3) that handle access decisions; namely *Policy Enforcement Point* (PEP), *Policy Administration Point* (PAP), *Policy Decision Point* (PDP), *Policy Information Point* (PIP), and a *Context Handler*.

[9] https://www.oasis-open.org/committees/download.php/2713/Brief_Introduction_to_XACML.html

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664



Source: OASIS[10]

Figure 3: XACML Flow & Architectural Components

Each of the depicted components has a certain purpose to serve:

- The *Policy Administration Point* (PAP) provides an interface or API to manage the policies (that are stored in a repository) and provides the policies to the *Policy Decision Point* (PDP).
- The *Policy Enforcement Point* (PEP) is the interface to the external world. It receives application-specific access requests and translates them to XACML access control requests. Subsequently, it denies or allows access, based on the result returned by PDP.
- The *Policy Decision Point* (PDP) is the decision point for access requests. It collects all necessary information from other actors and yields a decision.

[10] https://www.researchgate.net/figure/XACML-context-and-data-flow-diagram-Committee-2013_fig4_269986577

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

- The *Context Handler* coordinates the attribute value retrieval between PDP and PIPs, as well as the flow of access requests and responses between PDP and PEPs.
- The *Policy Information Point* (PIP) is where the necessary attributes for the policy evaluation are retrieved from several external or internal sources, such as the resource being accessed, the environment (for example, the time access request received), the subjects and so forth.

For more information, the reader may refer to the OASIS XACML web page[11].

Since XACML introduction (around 2003), several compliant tools, libraries, and frameworks have been developed and offered, both as free/open source software as well as commercial products. Some of the most well-known Java-based, open source tools are given next in Table 3. The main aspects considered in this table are the current version of each product (implying its maturity), its licensing model, and when its latest stable version has been released (indicating if the product is still being supported and maintained). Furthermore, brief comments have been added to highlight important facts and advantages or disadvantages.

Table 3: Java-based open source XACML tools

Product/Vendor	XACML Version	License	Latest release	Notes
Balana (library) WSO2 (https://github.com/wso2/balana)	3.0, 2.0, 1.x	Apache 2.0	Mar 2018	Based on Sun's XACML Implementation Seems to be the most used XACML implementation Previous experience from PaaSword project exists
Authzforce CE Thales & OW2 (https://github.com/authzforce)	3.0	Apache 2.0	Apr 2018	Lack of clear documentation for developing extensions
Picketbox JBoss (http://picketbox.jboss.org/)	2.0	LGPL 2.1	Feb. 2011	Merged with Keycloak project since 2015

[11] http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Xacml4j (https://github.com/xacml4j-opensource/xacml4j.github.io)	3.0, 2.0	GPL 3.0	Jul. 2014	No activity in project codebase at Github since 2014
XACML Light (http://xacmllight.sourceforge.net/)	2.0	Unknown	Unknown	PDP & PAP only
Heras AF University of Applied science Rapperswil, Switzerland (https://bitbucket.org/herasaf/herasaf-xacml-core)	2.0	Apache 2.0	Aug. 2016	It is an XACML 2.0 implementation
OpenAZ Apache Incubator (http://incubator.apache.org/projects/openaz.html)	3.0	Apache 2.0	n/a	Retired since Aug 2016
Sun's XACML Sun Microsystems Inc (http://sunxacml.sourceforge.net/)	2.0, 1.x	Open source	Dec. 2010	Too old. No active support anymore

Based on the information included in Table 3 (and especially in the last column), we opted to use WSO2 Balana engine for XACML 3.0 (latest). However, replacing it with another alternative is expected to be a relatively straightforward task, since the XACML policy engine resides inside the PDP component of the XACML architecture. Furthermore, the pluggable design of the server will allow easy replacement of plugins pertaining to the specific policy engine with new ones.

5.3.3 Use of Aspect and Aspect-Oriented Programming

Aspect-Oriented Programming (AOP) is a programming approach for software modularisation and separation of cross-cutting concerns [10]. This is achieved by adding extra functionality (called *Advice*) to existing code without modifying the source code.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

This addition typically occurs during the software building phase in a task called *weaving*, which is undertaken by specialised tools called *weavers*. The code to be modified is identified via *pointcuts*, which are specifications of those code artifacts (typically class and method signatures) needing to be enhanced with advices. *Pointcuts* can be external to the code or embedded as code metadata. *Pointcuts* also support specific query expressions for matching the relevant code. An *advice*, along with the *pointcuts* that specify the code it must be applied onto, is called an *Aspect*.

AOP allows the non-core functionality of a software component (for instance, logging of code executions, measuring duration, and authentication/authorisation) to be moved away from the code implementing the core business of the component. The non-core functionality is added and interleaved with the core functionality during the software build phase (via weaving). Thus, AOP enables the modularisation of functionalities into isolated (at source code-level) modules; this is usually referred as separation of concerns. Figure 4 depicts this concept; Methods A, B, and C implement the business logic, whereas logging, performance tracing and authorization are implemented separately (from Methods A, B, and C) and are weaved with them at compile time.

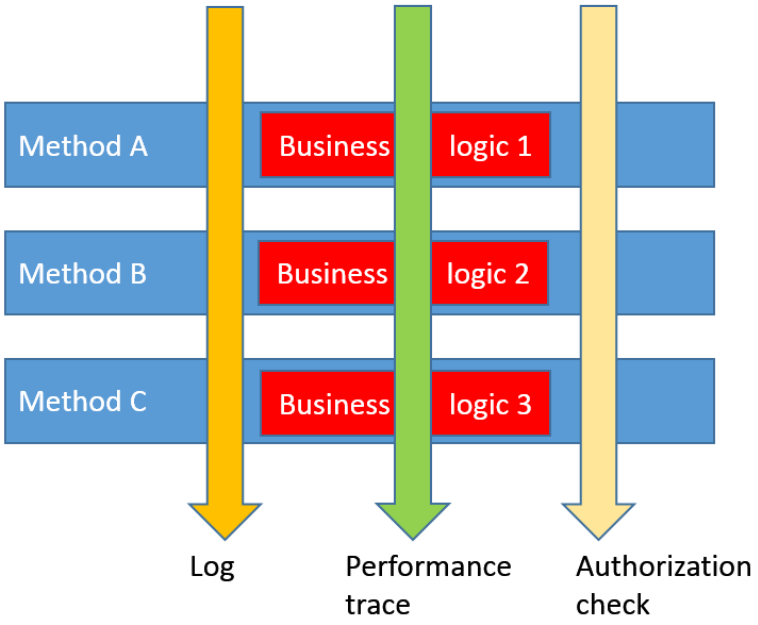


Figure 4: Aspect-Oriented Programming (Source [12])

Using this approach, the code implementing the core functionality is not cluttered with code related to other concerns. AOP provides a generic mechanism of code enhancement and extension which requires minimal or none at all modification of core code (depending on the AOP framework used).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

The Spring framework provides an AOP implementation[12]. Spring AOP is proxy-based, meaning each code artifact that can be enhanced with *advices* will be wrapped by a suitable proxy object that is actually invoked by the calling code. The proxy can subsequently pass control to the actually requested code. Proxies are automatically introduced at code-level (during weaving), while source code remains intact. Thus, this process is transparent to the programmer.

Regarding the use of *Aspects* in the *Authorisation Service*, an authorisation aspect will be introduced. The corresponding *advice* (i.e., the wrapping proxy code) will intercept the code invocation in order to perform a series of authorisation related tasks; namely, (a) create/reuse a PEP client object, (b) collect invocation information (i.e. method signature and arguments), (c) connect to a PDP server and pass the collected information, (d) receive the PDP server response (permit, deny, error), and (e) in case of permit (subsequently) call the actual (wrapped) code, or raise an authorisation error, otherwise. In the case where the wrapped code is a method of a Web or REST controller class, and that method is mapped to a Web or REST URL, then the corresponding (HTTP) request object is introspected to extract all HTTP related information.

5.3.4 Use of Request Interceptor for Spring-boot based components

Most Melodic Upperware components have been implemented as Spring-boot web applications. This means that they embed a minimal Tomcat server in order to accept incoming (HTTP) requests from other Melodic platform components, providing suitable REST APIs. The code implementing the REST API and receiving the requests needs to be protected with the Melodic Security services including the *Authorisation Service*.

One method for introducing the needed authorisation capabilities is by using Spring AOP, as has been explained in subsection 5.3.3 above. An alternative approach is by configuring the embedded Tomcat server (of the Spring-boot framework) to intercept the incoming requests and pre-process them before they actually reach the code that serves them. This is a standard step in the Tomcat HTTP request processing cycle and is implemented by adding special filters called interceptors. Interceptors can be added in Tomcat programmatically, during server initialisation.

Figure 5 depicts the interception process of an HTTP request by a Login Interceptor. The interceptor is invoked three times: (a) Pre-Handle: before calling the code that is meant to service the request (i.e. MainController), (b) Post-Handle: after the MainController returns and before rendering the response, and (c) After-Completion: when response has been sent back to the requestor.

[12] https://docs.spring.io/spring/docs/2.5.x/reference/aop.html

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

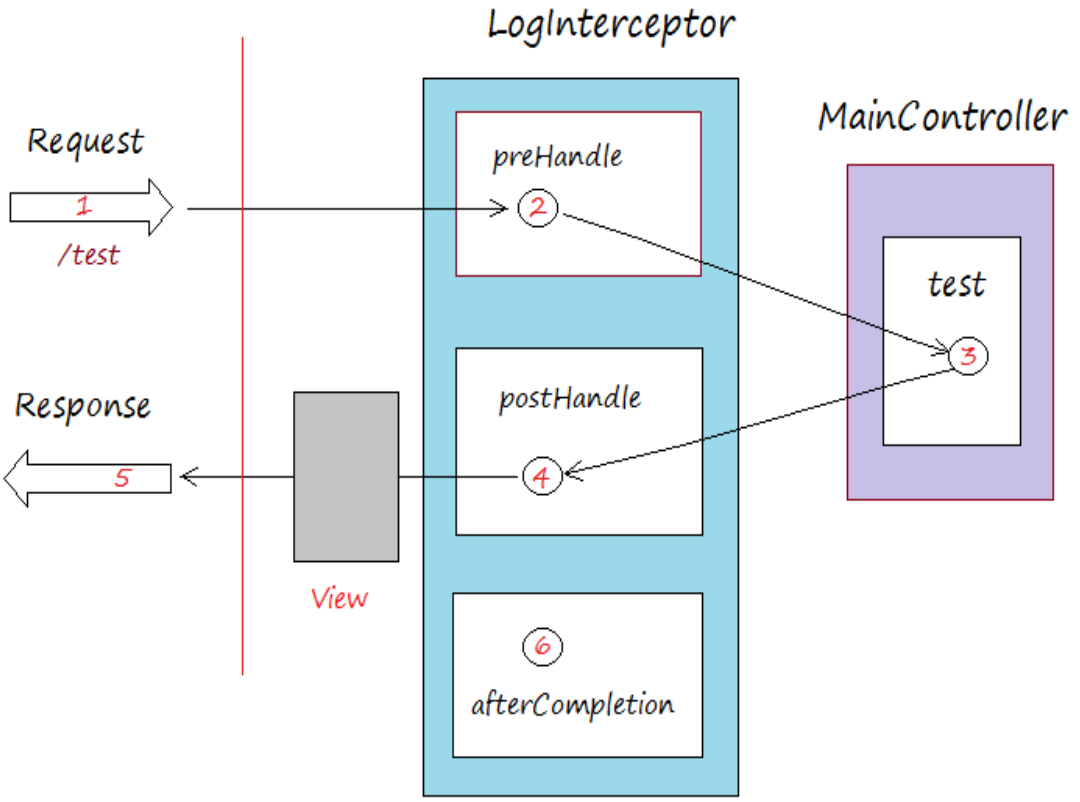


Figure 5: Tomcat request processing cycle[13]

Regarding Spring-boot web applications, interceptors can be added using an application configuration class that implements the WebMvcConfigurer interface. There, all needed interceptors can be added in the Tomcat interceptor registry, before the server starts.

This approach does not require any modification of application source code. Instead, a new configuration class can be written to configure an authorisation interceptor. This class must be packaged with existing code, and Spring-boot will take care of using it at runtime. The downside of this method is that it applies only to Spring-boot web applications with Tomcat server (Jetty is also possible). However, most Upperware components are as such.

5.4 Authorisation Service Architecture

In the remaining subsections, the architecture and operation of the Melodic Authorisation Service will be presented. This architecture follows the design decisions discussed in section 5.3 to fulfil the requirements of section 5.1.

[13] https://o7planning.org/en/11689/spring-boot-interceptors-tutorial

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

5.4.1 Attributes in Authorisation Service

The attributes handled by the Authorisation Service can be of three types; (a) access request-related attributes (e.g., requestor id, resource id), (b) request context attributes (not stated in the access request, but acquired from other sources) (e.g., requestor location and device), and (c) environment/platform-related context attributes (not pertaining to a specific access request. E.g., operational status of a platform component). The difference in the context in the two latter cases is that request context becomes invalid when the request has been processed, whereas environment/platform context evolves independently of the access requests. Figure 6 gives a high-level picture of the attribute flow in the *Authorisation Service*.

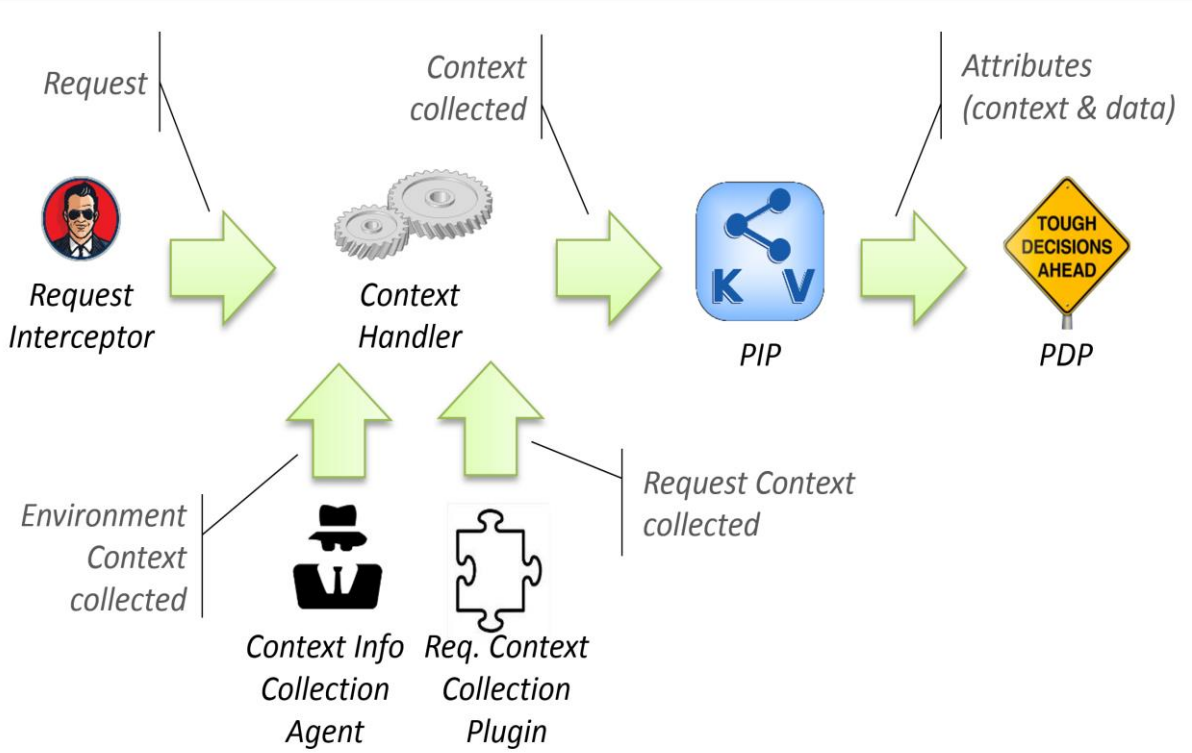


Figure 6: Attribute flow in Melodic Authorisation service

5.4.2 Authorisation Service Architecture

Figure 7 depicts the architecture of Melodic's *Authorisation Service*. The server part of the service in enclosed in a dashed box colored cyan. The main elements of the architecture is further explained after the figure.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

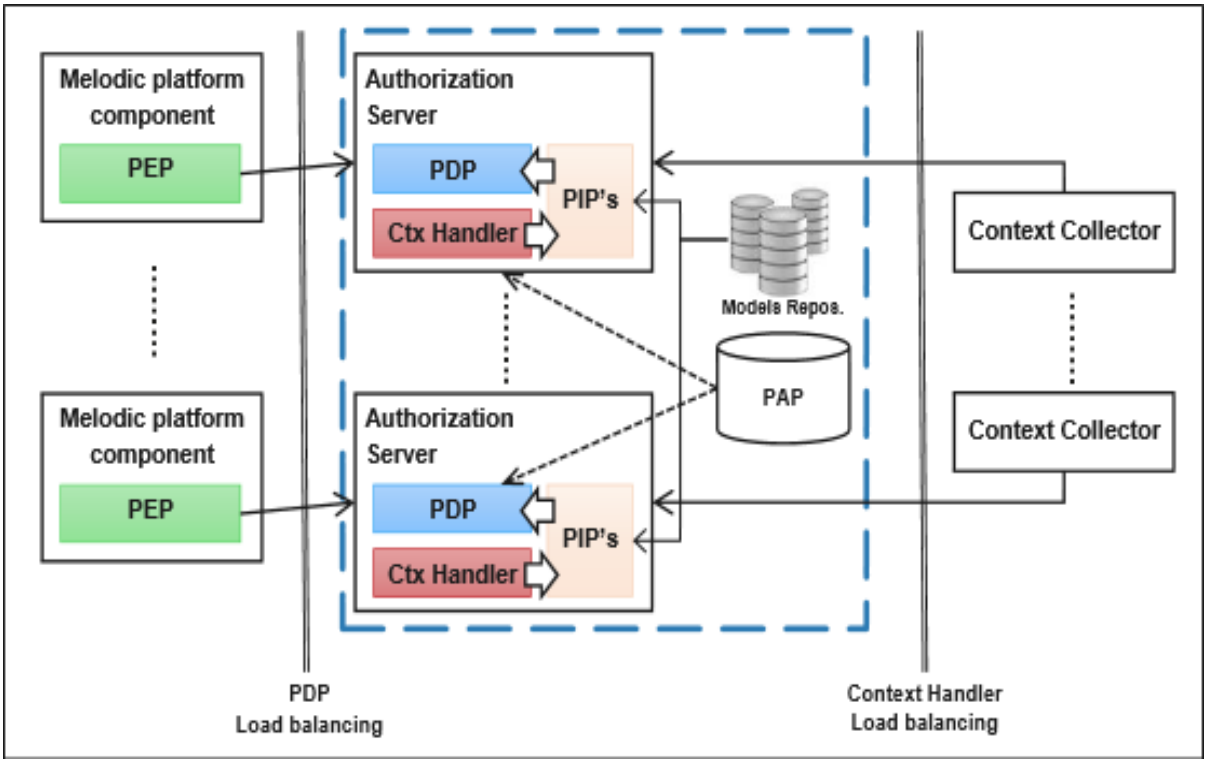


Figure 7: Authorisation Service architecture

- **Policy Enforcement Point (PEP).** It is embedded within the Melodic platform components that must be protected. This is where incoming access requests to resources enter the platform. PEP intercepts requests and interrupts the normal request flow, extracts request information and then contacts the *Authorisation Server* passing the extracted information. If the server returns a positive decision, the standard access request processing flow resumes. Otherwise, an error is reported, and the access is prevented. PEP is provided as an authorisation service client library, which is embedded in the platform components being guarded. Communication with PDP is achieved using the REST API exposed by the *Authorisation Server*, over an encrypted TLS connection.
- **Policy Decision Point (PDP).** It is a web service providing a RESTful API for receiving access request information from PEPs, evaluating them against policies and eventually authorising or declining access request. For this purpose, PDP contains a policy evaluation engine, namely WSO2 Balana. Upon configuration, PDP will first invoke *Context Handler* to collect additional (contextual) information from the request or the environment, and then evaluate the incoming request against policies. Several PDP nodes may coexist in a cluster to achieve high availability, fault tolerance and fast response times. Typically, all PDP nodes share the same configuration and the same policy repository.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

- **PDP Load-Balancing.** Conceptually, it stands between PEP clients and the PDP nodes. It is implemented at PEP-side as a configured list of PDP endpoints that are contacted either in successive order (round-robin) or selected randomly. Moreover, it is also feasible to add (third-party) HTTP proxy or load-balance server(s) and configure the PEP clients contacting it. The server(s) will in turn dispatch requests to PDP cluster nodes.
- **Context Handler (CH).** It is a web-service embedded in the *Authorisation Server*. Upon activation, it invokes the configured plugins to collect additional information (as attributes) about the context of the request. This contextual information is subsequently stored in a PIP (see below) in order to become available during policy evaluation. Furthermore, the *Context Handler* receives platform or environment-related context from *Context Collector*s (see below).
- **Policy Administration Point (PAP).** It is a simple PAP implemented as a directory containing the authorisation policies as a set of XACML files. Since authorisation service follows a centralised architecture, policies can be stored in a shared place accessible by all PDPs. Therefore, this simple implementation approach is adequate.
- **Policy Information Point (PIP).** The policy evaluation engine in a PDP, while processing a request, might require attributes not contained in the request itself. In this case, it invokes PIP plugins to retrieve the needed attributes. In XACML they take the form of key-value pairs, where keys can be any valid Uniform Resource Name (URN)[14]. PIPs are configured as plugins in the PDP configuration.
- **Context Collector (CC).** Context collectors are applications (or parts of applications) independent of the authorisation service, aiming at continuously collect information about the Melodic platform and its environment, and forward it to the *Context Handler*. It is expected that different context information, and thus context collectors, will be needed in different deployments of the Melodic platform.

A mapping of the architecture above onto the Melodic platform architecture is given next. Figure 8 gives the high-level architecture of the Melodic platform, where platform components protected by Policy Enforcement Points are suitably marked with a "security agent" figure. More information on the Melodic platform architecture can be found in deliverable D2.2 [1].

[14] *Uniform Resource Name (URN)* is a type of *Uniform Resource Identifier (URI)* used to identify resources within specfic namespaces

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

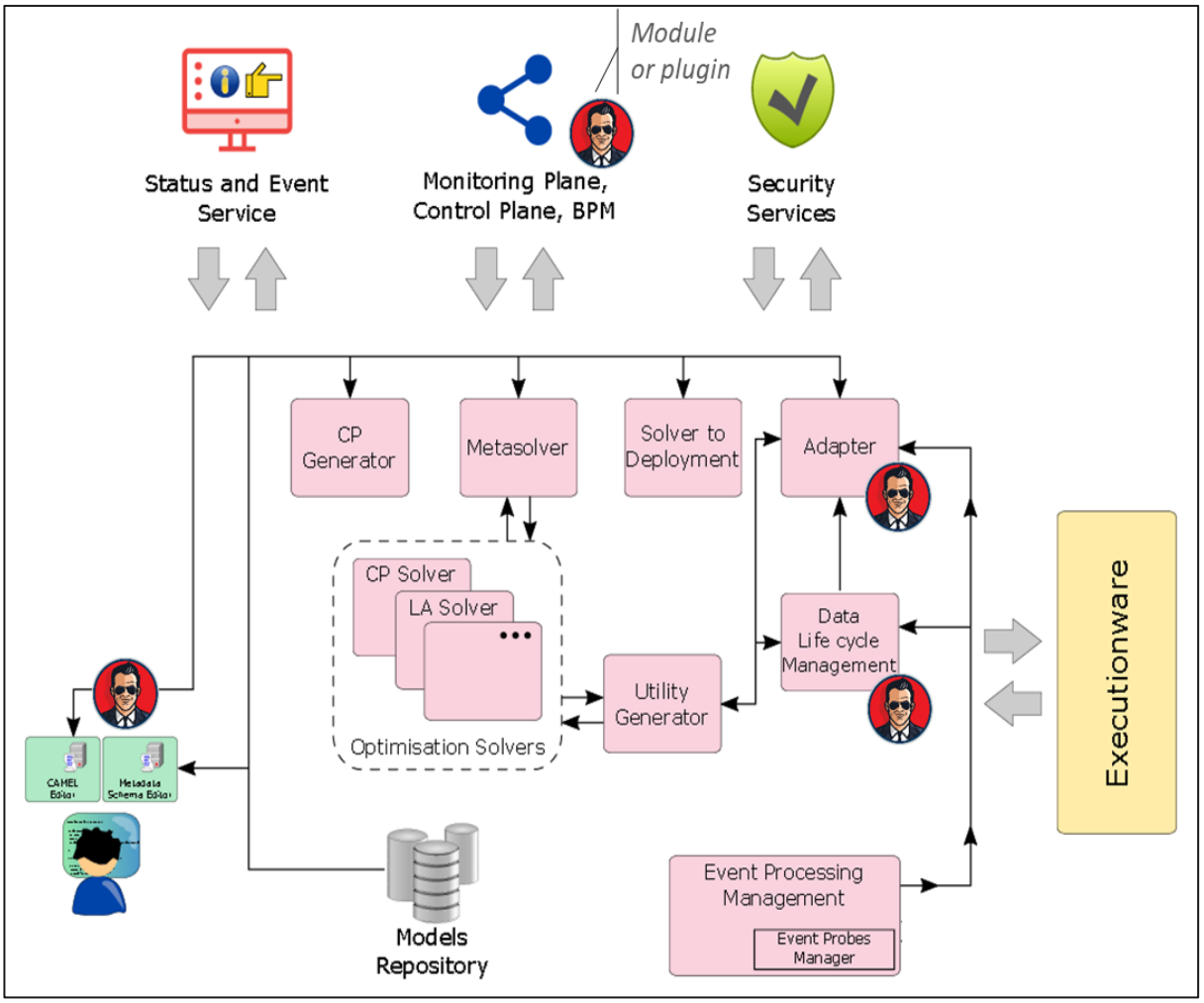


Figure 8: Authorisation service within Melodic platform architecture

As shown in Figure 8 above, the following platform components are protected with PEPs:

- **Business Process Management (BPM).** It coordinates the Upperware components and executes the workflow to generate and execute an application deployment plan out of a CAMEL model. When necessary, it also repeats the whole process or parts of it to introduce deployment plan updates, as a response to changes in application demands or environment. For more information, please refer to deliverable D2.2 [1], chapter 2, "Architecture Overview."
  During its operation, BPM contacts and is contacted by other Upperware components. A PEP client has been embedded in BPM, in order to protect it as well as other Upperware parts from a potentially compromised or malfunctioning component, or from outside-world interactions. PEP examines the origin and timeliness of the requests (which in this context are called from Upperware components) and authorises them.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

- **Adapter.** It is an Upperware component responsible for taking an application deployment plan and executing it by providing specific instructions to the Executionware. In order to verify that a given deployment plan conforms to the application deployment policies, a pre-authorisation step is taken. The plan parameters are checked against the relevant policies, and if rendered as conformant, the deployment starts. For this reason, the Adapter uses a PEP client to contact PDP to evaluate the plan against the posed policies. Plan pre-authorisation policies are different from access authorisation policies used for checking the access to previous components.
- **Data Lifecycle Management (DLM).** Similarly, to Adapter, the DLM system can also check a data placement and migration plan against relevant policies. For this reason, it also includes a PEP client.
- **Metadata Schema Editor (MuSE).** It is used to create and maintain the Melodic Metadata Schema and subsequently store it in the Melodic Model repository. MuSE comprises two layers; the User Interface layer, which executes in user browser, and the Backend, metadata management layer. The latter one also communicates and interacts with the Models repository. For this reason, the second layer includes a PEP client to protect itself from unauthorised access to its functionality and data.

6 User Authentication Service

This chapter documents the *User Authentication Service*. The User Authentication Service is an important component of the Melodic platform, as it allows, together with the authorisation service, to control access to the Melodic platform system. In Melodic, the User Authentication Service is based on actual standards in cloud application security: SAML2 (Security Assertion Markup Language) and OAuth for authentication.

This chapter is structured as follows. In subsection 6.1 the requirements for the User Authentication Service are extracted on the basis of the feedback from the project PaaSage, which is the underlying framework in Melodic. Subsection 6.2 gives a brief overview of the most frequently used user authentication models introduced in the literature. Based on the information of the first two subsections, the User Authentication Service design decisions are supplied in subsection 6.3.

6.1 User Authentication Service Requirements

The Melodic platform should use a unified method for user and component authentication. Based on the input provided in chapter 3, the summary of the

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

requirements related to the User Authentication Service (which covers users and components authentication) is presented in Table 4 below.

Table 4: User authentication security requirements

Req. id	Name of requirement	Requirements short description	Priority
1.	Unified method of user and component authentication.	Ability to use a unified way of user and component authentication. Preferably the same method should be used in a transparent way.	High
2.	User and components credentials are stored in one place only.	User and component credentials should be stored in one place in encrypted form, i.e., in only one component, which is responsible for authentication and thus should have access to these credentials.	High
3.	Token based authentication.	Authentication of user and method invocation should be based on generated tokens with an expiration timeout.	High
4.	Use industry standards for authentication.	For the authentication, industry standards (proven and verified) should be used.	Medium

6.2 Related work on user authentication

Based on [12], the evolution of the authentication methods for modern, cloud-based distributed applications is briefly presented in this subsection.

6.2.1 Monolithic applications

In a traditional monolithic architecture, users' requests are handled within a single process in the backend. A filter in the system boundary verifies the identity and access as well as determines the response or distribution of the request. As HTTP is a stateless protocol, it is usually based on a session that the server generates for the client to manage the user status. Here is the session control process:

1. The client provides his/her authentication credentials.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

2. The server validates the credentials. Depending on the validation results, we have the following alternative cases:
    a. Re-certification is performed if verification fails.
    b. The session is generated if verification is successful.
3. The client requests the resource within the session.
4. The server gets the session of the user by session id and response resource if the user has access.

The advantages of the session-based system are that it is simple and easy to implement while it allows for imposing more limitations in accessing the target system. However, it also suffers from many drawbacks as well. The server needs to save the session in memory, which may cause high memory usage and reduced performance. Moreover, the authentication feature is mixed with other systems' features together resulting in reduced system scalability and flexibility. As the traffic increases, the system needs to deploy multiple nodes to balance the respective load. Sharing sessions in multiple nodes is also a major issue. Besides, the session-based system uses cookies most of the time, so it should be able to deal with some cookie-based attacks from the side of the client.

6.2.2 Distributed Session Management

As the features of the system become more complex and the number of users increases, applications deployed on a single machine do not have enough resources to handle the user load. Moving from a single node to a cluster, the multiple nodes of the cluster must share a session when they use session-based authentication mechanisms. The following distribution solutions can be used for this purpose:

- A *sticky session* ensures that all the subsequent requests, constituting a request sequence along with the initial one, will be sent to the server that handled the first request in the sequence.
- *Session replication* means that each server saves session data and synchronizes through the network. So, it could be affected by network problems.
- *Centralised management* adds a specific server to manage all sessions. Every service request then maps to a session generated by the session server.

In any case, distributed sessions are complex in design and difficult to maintain.

6.2.3 Token-Based Authentication

A token-based authentication system allows users to enter their username and password in order to obtain a token which allows them to fetch one or more resources without using their username and password any more. Once their token has been obtained, the users can exploit it to have access to specific resources for a certain period of time. Figure 9 shows the process of token-based authentication. Through the use of a

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

token, there is no need to keep the session stored; the token is a self-contained entity that conveys all the user information. In addition, the token is stateless which makes it easy to deal with server-side scalability. The token can be adapted to different clients, such as browsers and mobile devices.

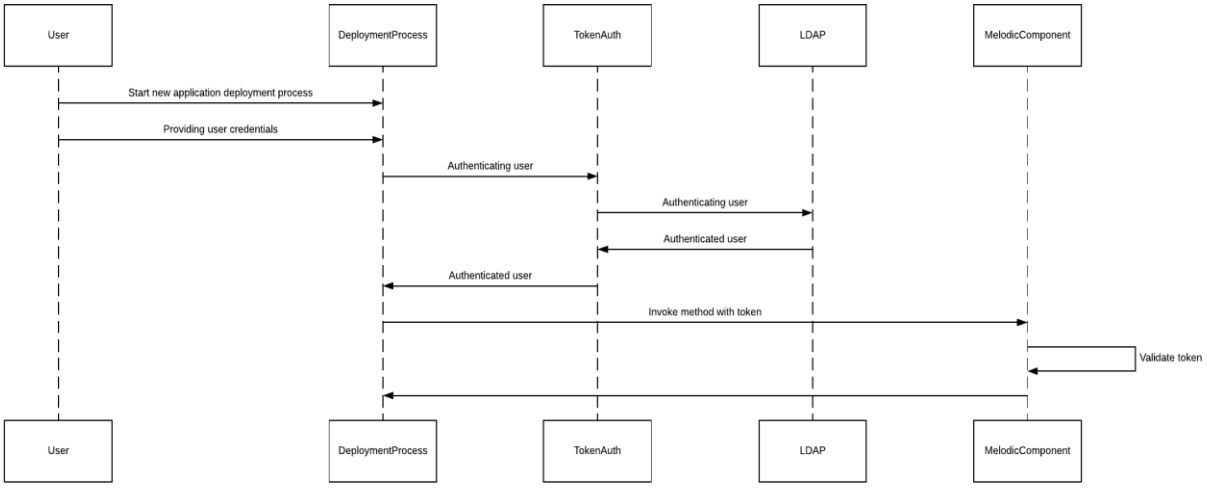


Figure 9: The authentication flow of operations

6.3 User Authentication Service Design Decisions

Using the requirements presented in subsection 6.1 and the information on user authentication work provided in subsection 6.2 as guidelines, we will subsequently supply the authentication service design decisions and provide brief justifications for our choices.

6.3.1 User credentials store

The LDAP server is chosen to be used as a store for user credentials. LDAP is a widely used standard for storing all user related information. In this respect, its usage allows for an easy and flexible integration of Melodic with organisational and enterprise security solutions.

6.3.2 Authentication mechanism – token-based

As presented in subsection 6.2, token-based authentication is recognized as a state-of-the-art mechanism and the most secure way of authenticating user and components in modern distributed applications. This is the reason for the usage of that method for the Melodic platform. The separate component *TokenAuth* (see Figure 9) is responsible for

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

generating tokens based on user credentials stored in LDAP. The tokens are generated based on user credentials, but used in each operations invocation between components.

6.3.3 Changes in process flow

The introduction of user and component authentication requires the following changes in the Melodic's deployment flow:

1. All method invocations should be executed in the context of a given user.
2. The authentication of method invocation will be based on the auth tokens generated by the *TokenAuth* component.

As a further extension, the usage of an access control service (described in chapter 5) as an authorisation mechanism for users and methods invocation is planned.

6.3.4 Changes in components' methods invocation

All Melodic components will be changed to use the authentication method based on tokens. For each method invocation, the token will be used. A token will be acquired once, at the beginning of the (deployment) process, from the *TokenAuth* component and will be used in all method invocations within the process. In case of expiration of the token, a new token should be generated.

6.4 User Authentication Service Architecture

The *User Authentication Service* architecture contains the following elements which implies changes to existing elements of the Melodic platform:

1. *LDAP Server* – an LDAP server with a data store for user credentials. The LDAP server is used to authenticate users based on provided credentials.
2. *TokenAuth* component – this component is responsible for issuing a JWT based token for an authenticated user. Token has an expiration time which is set based on the respective configuration in the system. Tokens are used to authenticate each method invocation on behalf of the user.
3. Changes in method invocation on the Melodic platform (between Melodic components) - each method invocation will use the user token to authenticate invocation of the particular method.

The authentication flow of operations is as follows:

1. User provides his/her username and password during the invocation of the deployment process on Melodic platform.
2. The user's credentials are used to authenticate the user in the *TokenAuth* component. This component attempts to validate the user in the LDAP Server;

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

upon a positive validation result, the token is issued and returned to the deployment process.

3. The issued token is used to invoke methods of Melodic's components. Only the supply of valid tokens during method invocation allows the actual execution of the given method.

Further extensions of the *Authentication Service* would be possible by using the *Authorisation Service* to control the access to the given method attempted to be invoked.

7 Cloud providers' credentials security

The management of cloud providers' credentials is crucial for the Melodic platform, as they allow accessing different cloud providers for supporting multi-cloud deployment. In Melodic, cloud providers' credentials are securely handled and passed through the deployment process without being stored in each component. They are only stored at one point, encrypted using the symmetric encryption algorithm AES with a key length of 256 bits, in the Cloudiator component. This supports the objective of achieving a high level of security for operations in a multi-cloud environment.

This chapter is structured as follows. In subsection 7.1, the requirements for the handling of cloud providers' credentials in a secure way are extracted on the basis of the feedback from the PaaSage project, which is the underlying framework in Melodic. Based on this information, the cloud providers' credentials security related design decisions are supplied in subsection 7.2.

7.1 Cloud providers' credentials security requirements

These requirements stem from the usage of the PaaSage framework. They are also based on common sense as well as general security principles. Based on the security requirements described in chapter 3, the summary of the requirements related to Cloud providers' credentials security is presented in Table 5 on the following page.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 5: Cloud providers' credentials security requirements

Req. id	Name of requirement	Requirements short description	Priority
1.	Encryption of cloud providers' credentials.	After users provide their cloud credentials, they should be stored in an encrypted form in the Melodic platform.	High
2 .	Storing cloud providers' credentials in one place.	Cloud providers' credentials should be stored in one place in the Melodic platform.	High
3.	Cloud providers' credentials encryption type.	Cloud providers' credentials should be stored in encrypted form using symmetrical encryption.	Medium
4.	Frequency of supply of cloud providers' credentials.	Cloud providers' credentials should be provided by the user only once.	Medium

7.2 Cloud providers' credentials security design decisions

Based on the presented requirements, the following design decisions have been taken:

1. Cloud providers' credentials will be stored only in the Cloudiator component (Executionware) as it is the sole component in the Melodic platform that requires them for execution deployment operations on the selected cloud providers. The cloud providers' credentials will not be stored and used in any other component of the Melodic platform. Storing in one place is an assumption in the logical architecture and it does not assume that it can't be replicated at the physical level, like database replication or file system synchronization.
2. Cloud providers' credentials will be encrypted using a symmetrical cryptography method – the AES algorithm with 256-bit key length.
3. The requirement 4 from Table 5 is not covered. It has medium priority and covering it could create another security leak.

The flow of cloud provider credentials are shown in Figure 10.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

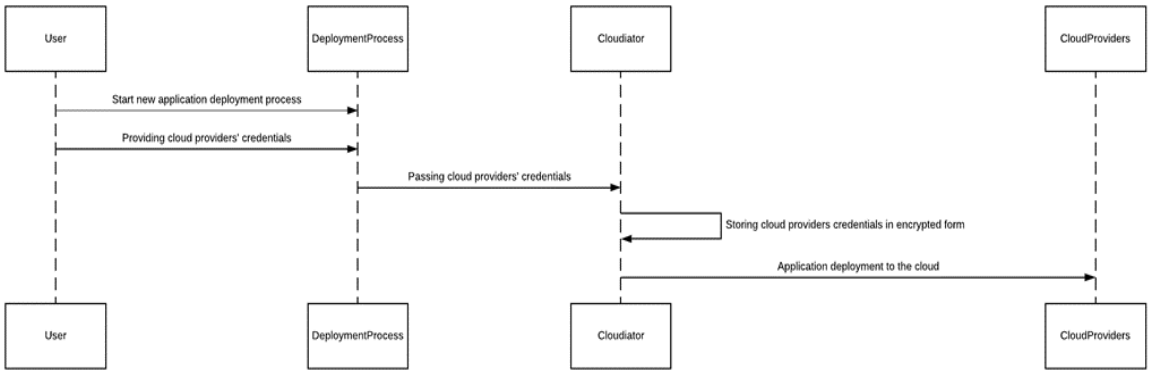


Figure 10: Cloud Providers credentials flow

8 Summary

Today, hardly anyone considers cloud solutions as a classic data centre infrastructure. It is also widely known that there are many applications, including those processing sensitive data, that turn many profits from safety, flexibility and economy of the cloud. This is possible thanks to security solutions implemented and applied by cloud providers.

The Melodic project delivers very specific types of security tools, corresponding to its character and objectives. Requirements were collected based on experiences from the PaaSage project, Description of Action for Melodic, requirements from use case applications and general security related experience. These security tools are described in this document. There have been three main solutions designed and implemented:

- Access control (chapter 5) – an innovative access control attribute-base model based on the XACML standard and the Balana solution, which enables advanced access control and authorisation in Melodic. It also allows the flexible definition of the security rules by using the XACML standard. It significantly increases the overall security level of the Melodic platform.
- User authentication (chapter 6) – it allows accessing and operating the whole system. In this project, the *User Authentication Service* is based on actual standards in cloud application security, i.e., the SAML2 and OAuth standards for authentication and authorisation.
- Cloud providers credentials security (chapter 7) – these credentials are crucial for the project as they allow access to various, independent cloud providers making Melodic a "multicloud" product.

The approach to the security aspects of cloud services described in this document ensures a significant level of security for the entire Melodic project and enables safe use of cloud solutions in general. Potential future works for Melodic security would be in direction of more tight integration between User Authentication Service and Authorisation Service.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

9 References

[1] Y. Verginadis, G. Horn, K. Kyriakos, F. Zahid, D. Baur, P. Skrzypek, D. Seybold, M. Prusiński and S. Mazumdar, D2.2 Architecture and Initial Feature Definitions, 2016.

[2] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith and P. Steggles, Towards a Better Understanding of Context and Context-Awareness., 2000.

[3] Y. Verginadis, I. Patiniotakis, C. Halaris, G. Mentzas, K. Kritikos and K. Jeffery, D2.4 Metadata Schema, 2017.

[4] Y. Verginadis, W. Żołnierowicz, P. Skrzypek, D. Seybold, K. Kritikos, S. Mazumdar, A. Schwichtenberg, F. Zahid, J. Domaschka, G. Horn, E. G. Gran, D. Baur, H. Masata and P. Góra, D2.1 "System Specification", 2016.

[5] J. Xu, Discretionary Access Control vs Mandatory Access Control.

[6] V. C. Hu, D. Ferraiolo, R. Kuhn , A. Schnitzer, K. Sandlin, R. Miller and K. Scarfone, Guide to Attribute Based Access Control (ABAC) Definition and Considerations, 2014.

[7] D. Ferraiolo and R. Kuhn, Role-Based Access Controls, 1992.

[8] D. Ferraiolo, R. Chandramouli, V. C. Hu and R. Kuhn , A Comparison of Attribute Based Access Control (ABAC) Standards for Data Service Applications: Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC), 2016.

[9] D. F. Ferraiolo , R. Chandramouli, D. R. Kuhn and C. Tong Hu, Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC), 2016.

[10] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier and J. Irwin, Aspect-oriented programming, 1997.

[11] J. Wang, An example to explain why we need AOP – Aspect Oriented Programming, 2016.

[12] M. Fowler and J. Lewis, Microservices. A definition of this new architectural term.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Appendix A: Assessing the Melodic Security Services via External Security Experts

According to the 1st Melodic review report, the project officer and the three project reviewers requested a further assessment of the security mechanisms designed and implemented in terms of the Melodic platform, via external security experts. In order to address this request, the consortium decided to seek for the appropriate external security experts that would be able to evaluate the current work with respect to the Melodic security services and potentially provide recommendations about their improvement, keeping in mind the context and scope of the Melodic Description of Actions (DoA). Based on this, the Melodic consortium decided to seek for two external security experts, one from the academic and one from the industry world, in order to achieve the necessary diversity of the external security audit's outcome. Thus, the external security experts used for this evaluation were the following:

- **Assistant Professor Antonis Michalas** from the Tampere University of Technology in Finland who also co-leads the Network and Information Security group (NISEC) of the university
- **SIDIO Sp. z o. o.**, [15]a Polish company comprising a team of practitioners and experts in the field of information and communication security. Specifically, two experts were involved from Sidio:
    - **Slawomir Kobus**, co-founder and Managing Director of SIDIO
    - **Adam Kuligowski**, co-founder of SIDIO

All these experts involved in the external evaluation of the Melodic security services bring several years of experience and a proven track record in cybersecurity as it is mentioned in the short bio sections provided in the two reports, available in the Appendices B and C of this deliverable.

Although the reader may find the details of these two reports at the respective appendices, in this new Appendix A of the deliverable we aim to summarize their findings, analyze their relation to the Melodic DoA and sketch the next steps with respect to the enhancements of the Melodic security services.

[15] From the industry, three companies were contacted and asked for an offer. SIDIO has been selected as the best value offer for the Melodic purposes.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Last, we note that the 2nd recommendation out of the 1st Melodic review report[16] is considered a part of the already planned WP6 work according to Melodic DoA. Specifically, the appropriate measurements and evaluations will be reported in deliverable D6.5 "Final Validation Results" (due M36) as part of the Melodic three use case demonstrators.

A.1 Summary of the external experts' assessment

In this section, we provide a summary of the external experts' assessment discerning the detected positive highlights and their recommendations for future enhancements of the Melodic security services.

A.1.1 Positive highlights

According to Prof. Michalas point of view, one of the strongest points of Melodic is that it has been designed by strictly following industrial standards. Specifically, the expert praised the decision to use SAML2.0 for authenticating and communicating attributes and privileges of users, as it is standardized, it is considered secure, it provides an excellent user experience, and it is supported by a big community that guarantees a satisfactory adoption of all the latest technological advancements. For the similar reasons the use of OAuth in Melodic was also considered as a very good idea since it is a popular authorization protocol following the token-based authentication which has the potential to provide tighter security. Another positive highlight was the adoption of the XACML standard and the way it was enhanced in Melodic for supporting and enforcing fine-grained, context-aware authorization. Apart from that, the expert praised the use of LDAP for storing users' credentials and the symmetric cipher AES-256 for encrypting cloud provider's credentials. In summary the expert stated that: "…by following industry standards, Melodic has the potential to support the latest technological advancements in the field of security. Therefore, and based on the fact that Melodic is still a research prototype the overall design is considered as a very good starting point that can be easily enhanced with extra security mechanisms".

Similarly, the report coming from the SIDIO experts highlights that all Melodic security services are very well architected using some of the most advanced security features. Specifically, the analysis conducted consisted in the assessment that Melodic and its particular mechanisms ensure satisfactory security levels in terms of the platform itself, its users, data and applications. Also, the key security functionalities and their

[16] *"Use quantitative measurements for assessing melodic performance and security against similar cloud solutions as this will raise trust in Melodic solution, - also for the three open source platforms that are integrated in Melodic, namely: PaaSage, CACTOS, and PaaSword."*

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

compliance with best practices was highly appreciated in terms of user credentials protection (based on AES-256 encryption), the token-based authentication of users and components (based on SAML 2 and JWT), the access control and pre-authorization (based on XACML), and the secure users' credentials storage (based on OpenLDAP). Among the noteworthy findings of the SIDIO experts were the following: "…We have found authorization services very advanced and well architected, as well as very flexible for the user of the platform".

A.1.2 Recommendations for enhancement

Both the external security experts' reports have provided a number of valuable recommendations that could be considered by the consortium for enhancing the Melodic security services. In this section, we have tried to aggregate the most significant details of these suggestions by compiling Table 6 and Table 7. There, we provide for each recommendation, a short description along with its prioritization and potential impact, the Melodic components that are affected, and last but not least an indication about the relevance of each suggestion to the Melodic DoA and an indication on whether or not it will be addressed in one of the upcoming releases of the Melodic platform.

Table 6: Consolidated details of the recommendations[17] provided by Prof. Michalas

Recom.	Short Description/Prioritization	Melodic Components affected	Impact *(if not supported)*	Relation to Melodic DoA *(Yes/No)* / Implementation in Melodic *(supported/to be (partially) supported/will not be supported)*
SR.01	Cloud providers that host data-aware applications could be running in a trusted state / COULD	-	Untrusted use of public cloud resources	**No / will not be supported**
SR.02a	Cloud providers that store users' internal data source can be protecting users' data from external attacks by encrypting the entire hard disks of the Cloud Service Provider / COULD	-	Untrusted use of public cloud resources	**No / will not be supported**

[17] We note that this table enhances the recommendations listed in chapter 4 of the security expert's report by including relevant suggestions clearly stated or implied in the previous chapters of the same report.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

SR.02b	The platform does not provide any mechanism to securely store and manage application's sensitive information (e.g. data storage credentials) /COULD	- DLMS - Cloudiator - Cloud providers' credentials security	Poor applications credentials management	**No / to be partially supported**
SR.03	Melodic should support SSL/TLS communication channels between all components / SHOULD	- All Melodic components	Lack of secure inter-components communication	**No / to be supported**
SR.04	Melodic should provide secure storage for intermediate and final results that are exported by the underlying components. Authorization should be applied for any access to read/write data to Models Repository / SHOULD	- CDO - Access control	Exposing CAMEL models / Maliciously amending CAMEL models	**Yes / to be supported**
SR.05	Melodic could provide mechanisms to enforce secure destruction of data, models and workloads that may contain sensetive information / COULD	- DLMS	Poor data sanitization support	**No / to be partially supported**
SR.06a	Melodic's token-based authentication system could protect users from impersonation attacks /COULD	- All Melodic components - User and component authenticatio n	Lack of token revocation or renewal issues that may affect all the components	**No / will not be supported**
SR.06b	The two layers of BPM and EPM should be able to authenticate each component that are interacting / COULD	- ESB, BPM - EPM - User and component authenticatio n	Untrusted communication between components	**No / to be partially supported**
SR.07	Melodic should provide access control for Melodic users and components / SHOULD	- Access control - CAMEL textual and a web-based editor - Metadata Schema Editor	Inappropriate users gaining access to CAMEL models	**Yes / supported**
SR.08	Melodic should support a mechanism for secret key revocation (and key rotation) of misbehaving platform administrators / SHOULD	- Cloudiator - Cloud credential's service	Poor credentials security (inability to revoke keys)	**No / to be supported**

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

SR.09	Melodic should support a mechanism for token revocation for compromised components / SHOULD	- All Melodic components - User and component authenticatio n	Lack of token revocation or renewal issues that may affect all the components	**No / to be supported**
SR.10a	Melodic should provide secure storage for cloud providers credentials and others (i.e. external storage credentials, database accounts) / SHOULD	- Cloudiator - Cloud providers' credentials security	Poor credentials security	**Yes / partially supported**
SR.10b	Explore other more sophisticated approaches (e.g. using a protocol based on hybrid encryption) / COULD	- Cloud providers' credentials security	Poor credentials security	**No / will not be supported**
SR.10c	Securely update the Cloud Providers credentials / SHOULD	- Cloudiator - Cloud providers' credentials security	Poor credentials security (inability to change credentials)	**Yes / to be supported**

Table 7: Consolidated details of the recommendations provided by the SIDIO experts

Recom.	Short Description / Prioritization	Melodic Components affected	Impact *(if not supported)*	Relation to Melodic DoA *(Yes/No)* / Implementation in Melodic *(supported/to be supported/will not be supported)*
Rec.1	Enabling modifications to credentials / HIGH	- Cloudiator - Cloud providers' credentials security	Missing cloud providers credentials update support	**Yes / to be supported**
Rec.2	Adding password complexity verification mechanism / HIGH	- Camel web editor - MUSE editor - BPM UI - User and component authentication	Platform vulnerable to brute-force attacks and unauthorized access	**No / to be supported**
Rec.3	Adding account lockout mechanism after x failed login attempts / HIGH	- Camel web editor - MUSE editor - BPM UI - User and component authentication	Platform vulnerable to brute-force attacks and unauthorized access	**No / to be supported**

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Rec.4	Introducing accountability of user activity by logging all events that make it possible to determine who, where and when introduced modifications affecting functionality or security / MEDIUM	- DLMS / Data Catalogue - User and component authentication	Lack of users' accountability. Platform vulnerable to brute-force attacks and unauthorized access	**No / to be partially supported**
Rec.5	Adding differentiating mechanism for levels of users' access to platform / LOW	- Camel web editor - MUSE editor - Access control	Lack of fine-grained access control	**Yes / supported**
Rec.6	Introducing two-factor authentication for Melodic users / MEDIUM	- Camel web editor - MUSE editor - BPM UI - User and component authentication	Poor overall platform security	**No / will not be supported**
Rec.7	Guaranteeing confidentiality of data transmitted via REST communication between Melodic and cloud environments / HIGH	- Inter-components Restful Communication	Lack of secure inter-components communication	**No / to be supported**
Rec.8	Providing security against Man-In-The-Middle (MITM) attacks for communication between Melodic and cloud environments. Introducing public key (certificate) verification process / MEDIUM	- Inter-components Restful Communication - Cloudiator-Cloud providers	Vulnerability to Man In The Middle (MITM) attacks	**No / to be partially supported**

Based on the analysis of the external experts' findings and recommendations, we identified a number of identical or similar recommendations among them that we highlight next. We also note the following mapping between severity and prioritization of these recommendations as follows:

- Could / Low
- Should / Medium
- Must / High

The identified similarities are provide in Table 8 on the following page.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 8: Related recommendations

Prof. Michalas	SIDIO
SR.10c/SHOULD	Rec.1/HIGH
SR.03/SHOULD	Rec.7/HIGH
SR.04/SHOULD SR.07/SHOULD	Rec.5/LOW
SR.06a/COULD SR.06b/COULD SR.09/SHOULD	Rec.8/HIGH
SR.02b/COULD SR.10a/SHOULD	-

A.2 Next steps for the Melodic security services enhancement

In this section, we list the experts' valuable recommendations (grouped by relevance) and discuss the next steps regarding the way that the Melodic consortium plans to address them or provides justification for some of them that will not be supported – all summerised in Table 9 below.

Table 9: Addressing the experts' recommendations

Recom.	Short Description / Prioritization	Discussion on the next steps
Rec.1	Enabling modifications to credentials / HIGH	We are considering a functionality (for the final Melodic platform release) that will allow the Melodic user to propagate any updates on the cloud providers' credentials to the Cloudiator.
SR.10c	Securely update the Cloud Providers credentials / SHOULD	
Rec.2	Adding password complexity verification mechanism / HIGH	Although this recommendation is out of the scope of the Melodic DoA, we are considering a functionality (for the final Melodic platform release) that will cater for the password complexity verification.
Rec.3	Adding account lockout mechanism after x failed login attempts / HIGH	Although this recommendation is out of the scope of the Melodic DoA, we are considering a functionality (for the final Melodic platform release) that based on a configurable number of failed logins the user account will be locked out.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Rec.4	Introducing accountability of user activity by logging all events that make it possible to determine who, where and when introduced modifications affecting functionality or security /MEDIUM	Although this recommendation is out of the scope of the Melodic DoA, we have already decided to develop and support a Data Catalog (for the final Melodic platform release) which will provide audit trails for the component actions. The Data Catalog will enable each of the Melodic platform components to log critical events pertaining the critical decisions they take with implications to the platform, as well as to the application deployments. In this way, this recommendation will be partially addressed since the focus of accountability and auditing is seen from the Melodic components point of view.
Rec.5	Adding differentiating mechanism for levels of users' access to platform /LOW	Rec.5 and SR.07 are already covered by the integration of the XACML-based Melodic authorization service to the Melodic editors and to the Adapter. SR.04 will also be addressed by enhancing the CDO model repository with access control capabilities (for the final Melodic platform release).
SR.04	Melodic should provide secure storage for intermediate and final results of that are exported by the underlying components. Authorization should be applied for any access to read/write data to Models Repository / SHOULD	
SR.07	Melodic should provide access control for Melodic users and components / SHOULD	
Rec.6	Introducing two-factor authentication for Melodic users / MEDIUM	This is a recommendation (prioritized as medium) not related to the Melodic DoA and it will not be addressed. The Melodic platform is designed to interface certain expert users (i.e. Admin, DevOps), but the implementation of two-factor authentication exceeds the scope of a research prototype.
Rec.7	Guaranteeing confidentiality of data transmitted via REST communication between Melodic and cloud environments / HIGH	Although the functionalities related to these recommendations were not mentioned in the Melodic DoA, we are planning to address them since they based on the experts' opinion seem critical for the security of the platform. Specifically, we are considering enhancing all the inter-components communication with an appropriate cryptographic protocol designed to provide communications security over unsecured networks (e.g. TLS, SSL). The communication between Melodic and the Cloud Providers is already based on signed certificates and TLS.
SR.03	Melodic should support SSL/TLS communication channels between all components / SHOULD	

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Rec.8	Providing security against Man-In-The-Middle (MITM) attacks for communication between Melodic and cloud environments. Introducing public key (certificate) verification process. / MEDIUM	The **Rec.8**, **SR.06a** and **SR.06b** are all related to the protection from the MITM attacks, a functionality not mentioned in the Melodic DoA. These recommendations have been characterized with medium priority by the experts, so it is not imperative to be fully addressed. Nevertheless, the **Rec.8** is already addressed with respect to the communication between Melodic (i.e. Cloudiator) and Cloud Providers based on TLS and certificates. **Rec.8** from the Melodic inter-component communications point of view (consequently relating to the **SR.06b**), it will be partially covered (for the final Melodic platform release) through the multi-way authentication support for two components between BPM and EPM layers, as a demonstration, with the use of certificates. This will be considered only for two components and for demonstration purposes, since it is expected to inject significant lag in the inter-components communication that may risk the reactivity of the platform. For a similar reason, **SR.06a** will not be addressed since the multi-tokens support for all Melodic components will normally be hosted on the same VM and in addition we consider that such a functionality exceeds the scope of this research prototype. Last, a token revocation process will be introduced (in the final Melodic platform release) that essentially addresses the recommendation **SR.09**.
SR.06a	Melodic's token-based authentication system could protect users from impersonation attacks / COULD	
SR.06b	The two layers of BPM and EPM should be able to authenticate each component that are interacting / COULD	
SR.09	Melodic should support a mechanism for token revocation for the compromised components / SHOULD	
SR.01	Cloud providers that host data-aware applications could be running in a trusted state / COULD	This recommendation is out of the scope of the Melodic DoA and it will not be addressed in terms of Melodic because it refers to external functionalities and services that may or may not be offered by the cloud providers.
SR.02a	Cloud providers that store users' internal data source can be protecting users' data from external attacks by encrypting the entire hard disks of the Cloud Service Provider / COULD	This recommendation is out of the scope of the Melodic DoA and it will not be addressed in terms of Melodic because it refers to external functionalities and services that may or may not be offered by the cloud providers.
SR.02b	The platform does not provide any mechanism to securely store and manage application's sensitive information (e.g. data storage credentials) / COULD	Although this recommendation is out of the scope of the Melodic DoA and it is characterised with medium priority by the expert, it will be partially addressed. We are considering a functionality (for the final Melodic platform release) that will enable a secure variable store for safely persisting such sensitive information within the platform (**see SR.10a**).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

SR.05	Melodic could provide mechanisms to enforce secure destruction of data, models and workloads that may contain sensitive information / COULD	Although this recommendation is out of the scope of the Melodic DoA and it is characterised with medium priority by the expert, it will be partially addressed. We are considering a functionality (for the final Melodic platform release) that will exploit the offered data sanitization support offered as a service by the used cloud providers (in case they support it). In this connection, a data life-cycle event management and triggering system, developed as part of the DLMS, can be exploited to execute data destruction/sanitization tasks upon resource decommissioning.
SR.08	Melodic should support a mechanism for secret key revocation (and key rotation) of the misbehaving platform administrators / SHOULD	Although this recommendation is out of the scope of the Melodic DoA and it is characterised with medium priority by the expert, it will be partially addressed. Specifically, we are considering a manual administrative procedure (for the final Melodic platform release) on revoking and re-issuing compromised keys (through appropriate scripts).
SR.10a	Melodic should provide secure storage for cloud providers credentials and others (i.e. external storage credentials, database accounts) / SHOULD	This is already partially supported as it was stated by the expert (cloud providers credentials are encrypted while at rest). This will be further enhanced (for the final Melodic platform release) as we address the SR.02b recommendation as stated above.
SR.10b	Explore other more sophisticated approaches (e.g. using a protocol based on hybrid encryption) / COULD	This recommendation is very interesting and forward looking, but it will not be addressed in terms of the Melodic research project because it refers to researching and implementing advanced encryption techniques which are out of the scope of the Melodic DoA. The use of hybrid encryption and other sophisticated approaches will be considered for implementation in Melodic as part of the after-project exploitation of the platform.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Appendix B:

Security Audit Report by Prof. Antonis Michalas

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

MELODIC: Multi-cloud Execution-ware for Large-scale Optimized Data-Intensive Computing

Security Analysis

H2020-ICT-2016-2017
Leadership in Enabling and Industrial Technologies;
Information and Communication Technologies

Prof. Antonis Michalas

November 10, 2018

Contents

1 Introduction **3**
1.1 Organization 3

2 Overview of the Melodic System and Current Security Mechanisms **5**
2.1 Interfaces to End Users 5
2.2 Upperware 6
2.3 Executionware 8
2.4 Auxiliary Services 9
2.4.1 Status and Event Service 9
2.4.2 Security Service 9
2.5 Control Plane and Monitoring Plane 10

3 Security Analysis of Melodic **11**
3.1 Melodic's Positive Highlights 13
3.2 Missing Security Functionality 14

4 Security Recommendations **16**

A Short Bio **19**

List of Tables

4.1 Identified Melodic Security Requirements 17

Chapter 1

Introduction

This document is a contribution to the H2020 Multi-cloud Execution-ware for Large-scale Optimized Data-Intensive Computing (Melodic) project that has been funded by the European Union under the H2020-ICT-2016-2017: Leadership in Enabling and Industrial Technologies; Information and Communication Technologies call.

This study has been conducted by external experts and **not** by the members of the current Melodic consortium[1] – however, proper channels of collaboration were established. The main aim of this study is to assess the overall security of the Melodic framework and provide some valuable insights regarding strengths and weaknesses of the current approach.

Before we proceed, it is worth mentioning that the current consortium already presented a well-rounded analysis on the security requirements of Melodic [7]. Our work extends this analysis by further analyzing the overall security of Melodic. As a result, we produce a list of security requirements that enhances the ones that has been already defined by Melodic's consortium.

Furthermore, the purpose of this document, and considering the scope of security services that was described in the DoW, is to provide the basis for the overall security design and functions that could be supported when/if Melodic is launched to the market. To this end, we propose a list of what we believe is considered as core security requirements for a framework like this. However, we leave the decision of adopting/implementing our suggestions to the discretion of the project's consortium.

1.1 Organization

The rest of the document is organized as follows:

- **Chapter 2 – Overview of Melodic System and Security:** This chapter, briefly describes the general architecture of Melodic along with its main components. Based

[1] This task is one of the requirements after the first review of the project that took place on the 6th of July, 2018 in Brussels.

on the described architecture we provide a list of main security points that we identified.

- **Chapter 3 – Security Analysis of Melodic:** In this chapter, we focus on assessing the security of Melodic. To do so, we first identify the existing security features of Melodic and then we proceed by highlighting potentially valuable security functionality that could enhance the platform beyond the lifecycle of the project and on the process of becoming from a research prototype a product.
- **Chapter 4 – Security Requirements:** This chapter contains a collection of the core security requirements that Melodic could consider in order to enhance its overall security.

Chapter 2

Overview of the Melodic System and Current Security Mechanisms

In this chapter, we present a high-level overview of Melodic architecture. This description is the coupled with a presentation of the identified critical security points that we will rely on in order to assess the overall security staus of Melodic. For a more detailed and well-rounded description of Melodic's architecture we refer reader to "D2.2: Architecture and Initial Feature Definitions" [8].

The Melodic framework (Multi-cloud Execution-ware for Large-scale Optimised Data-Intensive Computing), aims to provide ease and optimization at data-aware application deployment *across geographically distributed and federated cloud environments*. It is constructed from the following main component groups:

- the Interfaces to End Users (see Section 2.1);
- the Upperware (see Section 2.2);
- the Executionware (see Section 2.3);
- and two auxiliary services as Status & Event Notification and Security Services (see Section 2.4).

2.1 Interfaces to End Users

This is the entry point to Melodic for end users wish to use the platform for modelling their data-aware applications as well as the underlying data. To do so, Melodic is using CAMEL – a domain-specific language that allows users to define multi-cloud placement requirements and constraints. Through the Melodic framework, users can also add/or define new extensions for the CAMEL language.

For the description of applications and data models, Melodic interface supports both a textual and a web-based editor. The defined models include all user requirements as well as constraints of applications and a wide variety of data-sets such as security requirements, organization models, cloud provider models, deployment requirements, scalability rules, and service-level objectives. For the definition of new extensions, Melodic enables

amendment/expansion on Requirement, Metric, Scalability, Location, Provider, Security sub-models through the web-based Metadata Schema editor.

Based on design of the component Interfaces, we provide a list of the main security points that we identified:

2.1.1 Communication between end users and the Interfaces should be protected. Without building proper encrypted communication channels, an adversary can perform a wide range of attacks that could allow him to maliciously amend model definitions in CAMEL – a malicious behavior that can have severe consequences.

2.1.2 The framework should provide authorization to restrict access rights to the platform resources. For instance, only users with suitable roles should be able to access the Metadata Schema editor;

2.1.3 If CAMEL is shared between multiple users, it should not contain any sensitive information such as data storage credentials. Instead, the platform should support a separate way for user to securely store their sensitive information.

2.2 Upperware

Upon receiving the application and data models in a form of CAMEL modelling language from a group of Interfaces, the Upperware component calculates the optimal data placements and application deployments in cross-cloud environments. Its calculation is not only based on the actual requirements and constraints of the CAMEL format but it also takes into consideration factors such as the performance, the current workload as well as the underlying cost of the required cloud resources.

The Upperware group consists of the following components:

- Models Repository stores the models generated by end users through the Interfaces. This component is built on top of the internal component CDO Server for the model storage while the rest of the components exploit CDO Client in order to load the corresponding models.
- CP Generator is responsible for generating constraint programming (CP) models. These models express constant equations, based on application and provider models that are described in CAMEL.
- Utility Generator assigns a utility value (i.e. the goodness) to a candidate deployment configuration from the solver based on reconfiguration assessments from the other two components of the Upperware group, Data Lifecycle Management System (DLMS) and the Adapter. Functionalities of the two components are described later in this section. The utility value lies in the interval $[-1, 1]$ and its purpose is to show to compare the effectiveness of two configurations. More precisely, when the utility value is positive means that the candidate configuration is better than the current baseline configuration while in the case of a negative value it means that the fresh configuration is worst and should be avoided.

- Metasolver orchestrates the solvers' operations and ranks their outputs based on the CP models defined by the CP Generator earlier and the application description in CAMEL. Then, it exports the most optimal solution that is stored in the Models Repository.
- CP (Constraint Programming) Solver solves a specific deployment reasoning/ optimization problem that is described by a CP model and has been retrieved from the Models Repository and/or a local file system. Then it stores the solution back to the CP model.
- LA (Learning Automata) Solver solves a constraint mapping problem based on the realization that the problem is stochastic.
- Solver-to-Deployment applies a certain solution from the solvers to the application model defined in CAMEL. More specifically, at first it retrieves models from the Models Repository, which includes a CAMEL model describing the user application and the CP model which contains the corresponding solution. As a next step, it identifies the number of instances of the application components, virtual machines and connections between them that are needed for generating the required provider-specific deployment model.
- Adapter is responsible for validating a new CAMEL deployment model regarding time and cost aspects. Furthermore, it provides a comparison between the evaluated model and the current model. In addition to that, it separates the new model into different and well-defined action tasks and guides the Executionware on how to execute each one of them. Finally, it enriches the CAMEL model with running execution context information.
- Event Processing Management synchronizes and orchestrates Event Processing Agents in a distributed network in order to detect situations where reconfiguration is needed .
- Event Probes Manager decides and instructs the Executionware to deploy new monitoring probes and configure them to collect monitoring data on status of application components and used cloud resources.
- Data Lifecycle Management System (DLMS) manages the life-cycle of the registered data sources. More precisely, DLMS is responsible for selecting optimal data placement, ensuring that user-defined data requirements have been properly addressed and finally estimating costs to their data transfer and access.

Based on design of the Upperware component, we identify the following security issues that Melodic should carefully consider:

2.2.1 The communication channels between components need to be protected. Currently, components communicate through the Enterprise Service Bus (ESB) architecture. As a result, protecting the communication between all the components and the ESB is of paramount importance. Without building proper encrypted channels, an adversary could access and/ or amend optimal deployment models, change or steal monitored data and in general disrupt the proper function of Melodic.

2.2.2 Assume that two components c_i and c_j communicate with each other. More precisely, we assume that c_i wishes to communicate with c_j (i.e. c_i initiates the communication) Then, c_j should be able to authenticate and validate the trustful state of c_i as well as the freshness and the integrity of the request. Hence, every time that a component receives a request from another component, proper security mechanisms need to be in place in order to verify the validity of the request and protect c_j from receiving and processing malicious requests.

2.2.3 Data that is stored in the Models Repository needs to be protected from unauthorized access. The first step would be to provide a fine-grained access control mechanism. As a second step, it would worth investigating the support of encryption. While symmetric encryption could be a good solution we need to have in mind that it is not easy for CDO to support encrypted models. In addition to that, the fact that these models are frequently updated implies that the time needed for updating data and in some cases implementing key rotation might be a real burden for the proper run of Melodic.

2.2.4 APIs invocation should be protected from unauthorized access and possible corrupted/malicious entities.

2.3 Executionware

After the conclusion on the optimal solutions by the Upperware, Executionware is responsible for implementing the actual cloud deployments. Moreover, it manages and orchestrates cloud resources while at the same time monitors the deployed applications.

The Executionware group consists of the following components:

- Cloud Orchestration is built upon Cloudiator – a tool that allows the orchestration of web applications, discover private cloud resources offerings and optimized virtual machine placements. In addition to that, Cloudiator's Monitoring Services are further extended to support the integration of the Upperware's Event Processing Agents. Cloudiator's interface is the main entry point for the interaction of Upperware with the Executionware. At first, the Upperware requests a new application deployment to the Executionware. An application described in CAMEL will be mapped to a Job in Cloudiator, each application component to a Task, and instances to Process Entities. Based on that, the Executionware provisions the required nodes and deploys the processes on such nodes. At the same time, the Upperware requests the deployment and configuration of Event Processing and Monitoring on the provisioned nodes. Then the Executionware notifies the Upperware about the monitored deployment state.

- Resource Management Framework is an extended layer of Cloudiator that supports automated discovery of cloud resource offerings and their actual provisioning across multiple cloud providers.

- Data Processing Layer is another extended layer of Cloudiator that is implemented using a modular-based architecture. The Data Processing Layer can support Apache

Spark clusters orchestration by implementing SparkAgents, or Hadoop MapReduce clusters orchestration via implementing MapReduceAgents.

Based on the current design of Executionware, we identify the following security issues that Melodic should carefully consider:

2.3.1 The cloud providers' credentials are stored in Cloudiator in an encrypted form using a symmetric cipher. The symmetric secret key that is used for the encryption is only known to the platform administrators. Although cloud credentials are protected by using encryption, it is not easy to update the corresponding secret keys and/ or credentials. The platform should support a seamless way to change/ revoke the secret key, update/ delete the cloud provider credentials, etc.

2.4 Auxiliary Services

The auxiliary services involve the Status & Event Service and the Security Service. The first service provides a notification mechanism for all other components while the second one provides a set of secure operations.

2.4.1 Status and Event Service

This component is implemented as an ESB service and is responsible for generating and managing specific notifications and regarding events and the status of an operation in Melodic. Furthermore, it encompasses status notifications of operations such as returning deployment/ reasoning status of a given application, uploading CAMEL models, starting reasoning process and starting a deployment process.

Based on design of the Status and Event Service component, we identify the following security issues that Melodic should carefully consider:

2.4.1.1 The underlying communication channels between the Status & Event Service and ESB should be protected. Without proper protection, an adversary could interfere to create false alarm or mix up event notifications which may lead to unavailability of the system.

2.4.2 Security Service

This service is responsible for authenticating and authorizing the actions of core components of Melodic. To this end, the security service is focusing on the problem of data placement, performed and allowed actions of the underlying applications while it also supports a security and access policy repository. As a first step, the security service receives authentication requests from other components and/or new configuration and deployment actions from the Upperware's Adapter through ESB. Based on the received requests, it performs a list of security checks and publishes relevant reports containing the actual outcome (i.e. permit or deny), via ESB to the corresponding components. Furthermore, it can also export output reports that can be stored in the Models Repository.

- Authentication service securely stores all credentials for accessing cloud providers services and verifying core components on their actions like making decisions, placing/ configuring applications, migrating data, commissioning/ decommissioning cloud resources. The authentication is token-based by using SAML [2] and LDAP [1].
- Authorization service relies on a predefined set of policies and available context information in order to take a decision on allowing or denying the execution of placement or perform a reconfiguration actions. Moreover, it supports Attribute-Based Access Control (ABAC) policies [9]. The authorization is using the XACML standard [4] and WSO2 Balana platform.

Based on design of Security Service, we identify the following security issues that Melodic should carefully consider:

2.4.2.1 Both the authentication and authorization mechanisms need to be enhanced in order to provide a better level of security. Details on the existing authentication and authorization services will be presented in Chapter 3.

2.5 Control Plane and Monitoring Plane

For the integration of components, Melodic relies on two main layers – the Control Plane and the Monitoring Plane. The first layer is responsible for controlling actions within a process. The message propagation between components is done based on an Enterprise Service Bus (ESB) architecture for while the process orchestration is taking place through a Business Process Management (BPM). The second layer is responsible for monitoring data using a queue-based message broker for the delivery of messages.

Based on the current functionalities of the two planes, we identify the following security points that Melodic should carefully consider::

2.5.1 Communication between components and between the two layers should be protected;

2.5.2 For additional protection at the potential production phase of Melodic, the two layers could be able to authenticate each component that are interacting with.

Chapter 3

Security Analysis of Melodic

The current security design of Melodic focuses on the following three aspects:

- Cloud provider credentials protection;
- User and component authentication;
- User access control authorization.

For more details on the existing security solutions we refer reader to the deliverable "D5.03: Security requirements and design" [7].

Cloud provider credentials are needed for the Cloudiator component in the Executionware in order to be able to send deployment requests to the corresponding cloud service providers. Therefore, Melodic stores the cloud service provider's credentials only in the Cloudiator component. By doing this, it avoids any unnecessary transferring of such credentials between components. In addition to that, the credentials are encrypted using the AES symmetric cipher with a secret key of 256-bit length. Moreover, the underlying secret key is only known to the platform administrators.

Authentication is JWT token-based by relying on SAML2 (Security Assertion Markup Language) and OAuth [5] standards. Users' credentials are stored in LDAP server while tokens are generated by the separate component TokenAuth. When a user wishes to deploy an application on the platform, she needs to provide a username and a password to the deployment process. This information is then sent to the TokenAuth component which connects to the LDAP server through which authenticates the user. As soon as the user's credentials are validated by the LDAP server, TokenAuth generates a unique token based on the received credentials. The generated token is then issued to the deployment process. Then, the token is used every time that a method is invoked between the platform components. The involved components are then responsible for validating the token in order to allow or deny the actual execution of the invoked method.

The platform provides two types of authorization. One is a pre-authorization phase that enforces a deployment and/or data placement plan to conform to the given set of policies, constraints and limitations (such as regulation, budget, resource, security, etc). The other type is the actual authorization phase which protects the platform resources (such

as services, components and workflows) from unauthorized access attempts or from compromised components that try to access the platform. The authorization method is based on the Attribute Based Access Control (ABAC) model for authorizing resource requests, and eXtensible Access Control Markup Language (XACML) for describing authorization policies, access control requirements, and querying access to resources. More precisely, the authorization process is implemented by five core components as follows:

- The Policy Enforcement Point (PEP) is responsible for receiving access requests and authorizing them based on the output that will be received by the Policy Decision Point (PDP). Furthernore, PEP is embedded in the following platform components:
  - Business Process Management (BPM);
  - Adapter;
  - Data Lifecycle Management System (DLMS);
  - Metadata Schema Editor (MuSE).

  In BPM, PEP examines the timeliness and origins of the requests received by the other Upperware components. If the requests are valid then PEP provides authorizes them. In Adapter, PEP is responsible for validating the application deployment plan against a set of defined policies prior to execution by the Executionware. In DLMS, PEP checks a data placement and migration plan against a set of defined policies. Finally, in MuSE, PEP authorizes any access to its functions as well as data.
- The Policy Administration Point (PAP) is responsible for managing policies as well as provides the Policy Decision Point (PDP) with policies.
- The Policy Decision Point (PDP), is implemented by the WSO2 Balana open source framework, retrieves requests from PEP and evaluates them based on a set of policies and necessary collected information that are received by PAP, PIP and the Context Handler. The main goal of PDP is to provide a decision in order to allow or deny a requested access.
- The Policy Information Point (PIP) stores the necessary attributes that are needed for a proper evaluation of policy. These attributes can be the of the access request, the corresponding resource id, etc. and can be based on data from both internal and external sources.
- The Context Handler collects additional attributes and information regarding the context of the received requests, context related to the underlying platform and environment that are subsequently stored in the PIP.

Communication between PEP and authorization server (which contains PDP, Context Handler and PIP) is *only* protected over SSL/TLS.

3.1 Melodic's Positive Highlights

Before describing the identified missing security functionality (Section 3.2) it is worth describing (briefly) the advantages and the strong points of the current Melodic architecture. One of the strongest points of Melodic is that it has been designed by strictly following industrial standards.

First of all, the decision to use SAML2.0 is considered as an excellent choice. SAML is an XML-based framework that is used to authorize, authenticate and communicate attributes and privileges of a user. It provides numerous benefits to enterprises, organizations and governments. However, SAML has been widely adopted for three primary reasons: is standardized, it is considered as secure, and it provides an excellent user experience. Apart from that, the fact that SAML is a widely used framework implies two important things:

- There is a big community that supports and further develops the framework in such a way that the satisfactory adoption of the latest technological advancements is "guaranteed".
- A possible integration of Melodic services with other services can be proved to be a relatively easier task than what it would have been if Melodic was based on a not so popular and widely used framework.

Apart from that, Melodic is also using OAuth. OAuth is a popular authorization protocol that enables applications to access HTTP services on behalf of users by enabling delegated tokens rather than the users' main credentials. Currently, OAuth is being developed and has the full support of the IETF OAuth working group. Integrating OAuth in Melodic is again a decision that can be proved to be wise in the future. This is not only due to the flexible token-based approach that the protocol is using but also due to the fact that OAuth is supported by many different providers and platforms. Therefore, integration and communication of Melodic with other services could be done in an easy and (possibly) straight forward way. Furthermore, using a token-based authentication instead of a traditional username and password approach reduces the burden and insecurity of repeating submitting users' credentials. Tokens are only valid for a limited (short) time – hence replay attacks can be avoided (in certain scenarios). Additionally, these tokens are both revocable and refreshable. Therefore, it is believed that token-based authentication has the potential to provide tighter security. Apart from that, the XACML (eXtensible Access Control Markup Language) standard that is used to enforce authorization policies supports defining fine-grained, attribute-based policies as well as role-based policies. Furthermore, it also supports conditional authorization, combination of policies, and conflict resolution. Relying on XACML standard provides lot of flexibility to the underlying services since it is considered to be independent of the implementation.

Apart from that, Lightweight Directory Access Protocol (LDAP) is utilized for storing users' credentials. LDAP is widely used and it is considered as a reliable central credentials storage mechanism that also provides easy accessibility. In addition to that, as LDAP supports TLS/SSL, users' credentials can be protected via a secure communication channel.

Finally, symmetric cipher AES with a secret key of 256-bit length is used to encrypt cloud provider's credentials. AES is considered as a very good and reliable choice since it is a well-known and standardized symmetric cryptosystem that is also being used in industry. Moreover, AES is also considered to be semantically secure – a very important property when you store sensitive information.

In summary, by following industry standards, Melodic has the potential to support the latest technological advancements in the field of security. Therefore, and based on the fact that Melodic is still a research prototype the overall design is considered as a very good starting point that can be easily enhanced with extra security mechanisms.

3.2 Missing Security Functionality

Although the cloud service provider's credentials are protected by using symmetric encryption – an approach that for the current status of Melodic is considered as adequate – it would be a good practice to also explore other more sophisticated approaches (e.g. using a protocol based on hybrid encryption) in case Melodic goes into the market. In addition to that, another point that needs some attention is that of key rotation. More precisely, the secret key needs to be changed regularly in order to enhance credentials' security and privacy. Moreover, every time that the key is changed, re-encryption of the underlying credentials is required. Apart from that, in order to update the cloud credentials, a similar process where the fresh credentials will be submitted and stored in an encrypted form is required. These last points are the first issues that Melodic's consortium will have to look at if they decide to further expand the existing security mechanisms. Providing a reliable and realistic solution for these problems can really pave the way for building a much stronger security model that will allow Melodic to be launched in the market.

In addition to that, it is quite common that data-aware applications require some type of sensitive information to run such as data storage credentials, database accounts, etc. For the sake of security, such sensitive information should *not* be hard-coded into the application's source code, or stored in any component as plaintext. Based on the current architecture of Melodic, *the platform does not provide any mechanism to securely store and manage application's sensitive information.*

Apart from that, the platform executes authentication based on SAML2 and OAuth standards. For each user, the component TokenAuth generates a token which could be used by other components for each method invocation. By relying on token with expiration time, the platform avoids requirement of repeated submitting users' credentials and enhance its security (provide at least a basic protection against replay attacks). However, the fact that *the same token is used by different components of the platform* leads to some disadvantages. For instance, when a token is revoked or renewed, its revocation or renewal needs to be propagated promptly to every component that is using it. Additionally, the token only contains user's specific information but it does *not* contain any information on the component to which it is issued. Instead, issuing different tokens to different components for the same user is a more secure and more complete approach for authentication. Considering the adoption of a Single-Sign-On scheme could be a solution to this problem.

Moreover, the platform supports authorization which involves pre-authorization enforcing a deployment and/or data placement plan to conform to given policies and au-

thorization protecting the platform resources from unauthorized access. More precisely, authorization support is evident for the appropriate components but it seems that it has not been implemented for one critical component of the architecture – the Models repository. Models Repository stores models that are generated by the end users, constraint programming (CP) models generated by CP Generator, optimal solutions calculated by Metasolver, security check reports outputted by security services, etc. Such models and information are vital for the proper function of the platform – hence the proper security of this information is of paramount importance. To this end, *authorization should be applied for any access to read/write data to Models Repository.*

Furthermore, the data stored in Models Repository are important for Melodic and they need to be protected. As a result, the first and most important measure is to protect the confidentiality of the stored data so that any corrupted entity will fail to extract any valuable information regarding the content of the stored data.

Finally, the communication between all components should be protected by (at least) enabling SSL/TLS. Currently, only the communication between the components of PEP (Adapter, Data Lifecycle Management System) and PDP in Authorization Sever is confirmed to be running over TLS/ SSL. However, *it is not clear how the rest of the communication channels are protected – especially between the ESB and other components.*

Chapter 4

Security Recommendations

In this chapter, we present a concrete list of the main security requirements that were exported from our analysis on the security of Melodic. For the evaluation of each recommendation we have followed the RFC 2119 convention [3]. In Table 4.1, we present the identified security recommendations that needs to be considered by the consortium of Melodic in case they decide to launch Melodic in the market (currently Melodic is a research prototype and therefore a subset of these recommendations may be applied).

ID	Recommended Security Requirements	Supported
SR.01	Cloud providers that host data-aware applications COULD be running in a trusted state (i.e. satisfying pre-defined security policies, and being launched by using a trusted launch protocol such as the one described in [6] and makes use of the Trusted Computing principles).	No
SR.02	Cloud providers that store users' internal data source CAN be protecting users' data from external attacks by encrypting the entire hard disks of the Cloud Service Provider.	No
SR.03	Melodic SHOULD support SSL/TLS communication channels between all components as well as between users and Melodic.	Partially
SR.04	Melodic SHOULD provide secure storage for intermediate and final results of that are exported by the underlying components.	Depends on authentication on CDO Server in Models Repository

SR.05	Melodic COULD provide mechanisms to enforce secure destruction of data, models and workloads that may contain sensitive information.	Partially through DLMS
SR.06	Melodic's token-based authentication system COULD protect users from impersonation attacks.	Not completely. A component may impersonate a user by reusing granted token.
SR.07	Melodic SHOULD provide access control for Melodic users and components.	Yes
SR.08	Melodic SHOULD support a mechanism for secret key revocation of the misbehaving platform administrators.	No
SR.09	Melodic SHOULD support a mechanism for token revocation for the compromised components.	No
SR.10	Melodic SHOULD provide secure storage for cloud providers credentials and others (i.e. external storage credentials, database accounts).	Partially – cloud providers credentials are encrypted while at rest.

Table 4.1: Identified Melodic Security Requirements

Bibliography

[1] B. Arkills. *LDAP Directories Explained: An Introduction and Analysis*. Addison-Wesley Professional, 2003.

[2] A. Armando, R. Carbone, L. Compagna, J. Cullar, and M. L. Tobarra. Formal analysis of saml 2.0 web browser single sign-on: breaking the saml-based single sign-on for google apps. In V. Shmatikov, editor, *FMSE*, pages 1–10. ACM, 2008.

[3] R. L. Barnes, S. T. Kent, and E. Rescorla. Further key words for use in rfcs to indicate requirement levels. *RFC*, 6919:1–6, 2013.

[4] D. Ferraiolo, R. Chandramouli, R. Kuhn, and V. Hu. Extensible access control markup language (xacml) and next generation access control (ngac). In *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control*, ABAC '16, pages 13–24, New York, NY, USA, 2016. ACM.

[5] D. Hardt. The oauth 2.0 authorization framework, 2013.

[6] N. Paladi, C. Gehrmann, and A. Michalas. Providing user security guarantees in public infrastructure clouds. *IEEE Transactions on Cloud Computing*, 5(3):405–419, July 2017.

[7] P. Skrzypek, Y. Verginadis, I. Patiniotakis, and C. Chalaris. H2020 melodic: D5.03 security requirements and design. Technical report, 7bulls, 2018.

[8] Y. Verginadis, G. Horn, K. Kritikos, F. Zahid, D. Baur, P. Skrzypek, D. Seybold, M. Prusiski, and S. Mazumdar. H2020 melodic: D2.2 architecture and initial feature definitions. Technical report, Simula Research Laboratory, 2018.

[9] E. Yuan and J. Tong. Attributed based access control (abac) for web services. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '05, pages 561–569, Washington, DC, USA, 2005. IEEE Computer Society.

Appendix A

Short Bio

Prof. Antonis Michalas received his PhD in Network Security from Aalborg University, Denmark and he currently works as an Assistant Professor at the Department of Pervasive Computing at Tampere University of Technology, Faculty of Computing and Electrical Engineering. Within the Department of Pervasive Computing there is the Network and Information Security group (NISEC) which is co-led by Antonis and Prof. Billy Bob Brumley. The group comprises PhD students, professors and researchers. Group members conduct research in areas spanning from the theoretical foundations of cryptography to the design and implementation of leading edge efficient and secure communication protocols. Apart from his research work at the NISEC group, as an assistant professor Antonis is actively involved in the teaching activities of the University. Finally, his role expands to student supervision and research projects coordination.

Prior to that, he was working as a lecturer (assistant professor) in Cyber Security at the University of Westminster, London. As a lecturer he was teaching both undergraduate and postgraduate courses related to cryptography, forensics, cyber security and network security. His role expanded to student supervision and research group coordination. More precisely, during his time at the University of Westminster, Antonis established his own research group (CSec). In parallel Antonis was an active member of the department's project development and research activities.

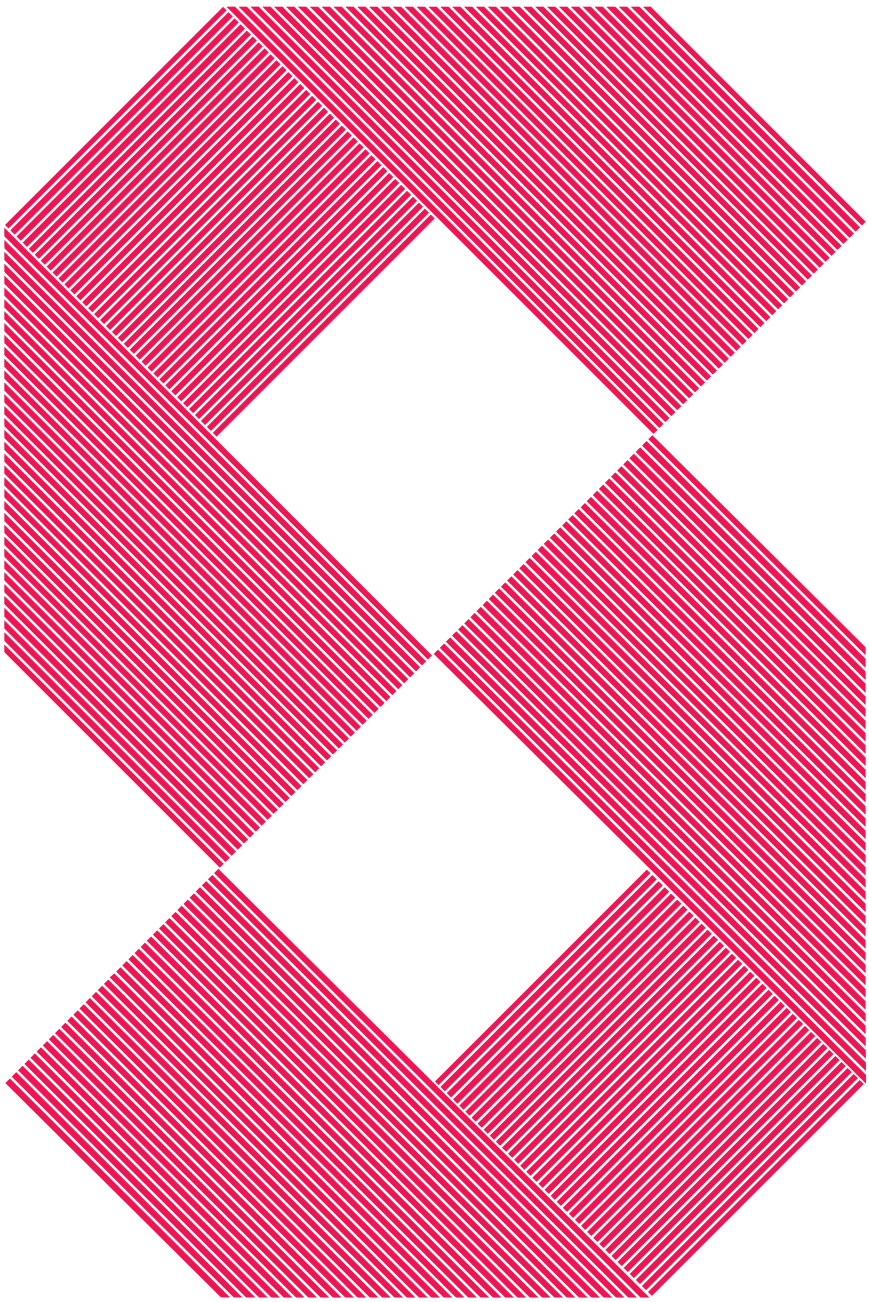
Earlier, Antonis was working as a postdoctoral researcher at the Security Lab of the Swedish Institute of Computer Science (SICS) in Stockholm, Sweden. As a postdoctoral researcher at the SEC Lab he was actively involved in national and European research projects and combined research with student supervision and project management.

Finally, Antonis has published a significant number of papers in field-related journals and conferences and has participated as a speaker in various conferences and workshops. His research interests include private and secure e-voting systems, reputation systems, privacy in decentralized environments, cloud computing, trusted computing and privacy preserving protocols in eHealth. More information on his profile can be found at: www.amichalas.com

Appendix C:

Security Audit Report by SIDIO Sp. z o.o.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664



SIDIO

CYBERSECURITY

Title:	Security Audit Report for Melodic Platform
Date:	05 November 2018
Author(s)	Adam Kuligowski, SIDIO Slawomir Kobus, SIDIO

TABLE OF CONTENTS

1. INTRODUCTION 4

2. EXECUTIVE SUMMARY 4

3. ANALYSIS AND CONCLUSIONS 5

3.1. Protection of user credentials when accessing cloud environment 5

3.2. Authentication of users and components 6

3.3. Authorization and access control 7

3.3.1. Access control 7

3.3.2. Pre-authorization 8

3.4. Storing users' credentials 9

3.5. Users accountability 9

3.6. Protection from unauthorized access to the user interface 10

3.7. Securing REST communication between Melodic and cloud environments 11

3.8. High Availability 12

4. RECOMMENDATIONS 13

5. ABOUT THE AUTHORS 14

5.1. Slawomir Kobus 14

5.2. Adam Kuligowski 15

5.3. SIDIO 15

1. INTRODUCTION

The purpose of this document is to present the results of the analysis of architecture and security mechanisms used in Melodic platform, to indicate key elements affecting security level of the whole platform and to assess them. Results of this analysis will determine the adequacy of technologies used and provide recommendations for improvements of the elements according to security requirements.

2. EXECUTIVE SUMMARY

Melodic is well architected in terms of security services, using some of the most advanced security features, especially in the authorization part. We have found authorization services very advanced and well architeced, as well as very flexible for the user of the platform.

Main findings for improvement regarding security of Melodic platform are related to user interface and accountability of user activity on the platform. They include among others: no password complexity verification mechanism, no account lockout mechanism due to using wrong password too many times (making the platform vulnerable to brute-force attacks), no user activity log (monitoring both login attempts and user activities after successful logging into the account).

There is also a need for implementation of two-factor authentication to access Melodic user interface. Two-factor authentication will significantly increase platform's level of security in case of sharing the interface via the Internet by platform administrators or in case of changing the model of cloud computing (to Platform-as-a-Service).

It has also been noted that the credentials (logins and passwords) provided by users to access cloud environments cannot be modified. It is however a functional, not security-related issue.

Vulnerability to Man In The Middle (MITM) attacks has been observed in terms of REST API communication between Melodic platform and cloud environments as well as the need to

implement verification mechanism for public keys (certificates) used as a part of such communication.

The assessment also noted lack of security mechanisms for applications deployed and maintained on Melodic. According to auditors it is not critical. However, a possibility of implementing such mechanisms by taking them into account when creating application models and integrating Melodic platform with specialized third-party security solutions (free or commercial), such as web application firewalls (WAF) or application delivery controllers (ADC) would be worth considering. Including application security mechanisms in created models and proper integration can prove Melodic's significant competitive advantage when compared to similar solutions.

Architecture of the platform, its particular components, models and implementation methods used have not raised any concerns.

3. ANALYSIS AND CONCLUSIONS

The analysis conducted consisted in assessment of the architecture of Melodic platform, its particular elements and mechanisms used to ensure satisfactory security level in terms of the platform itself, its users, data and applications. Key security functionalities and their compliance with best practices in a given sector have also been taken into account.

3.1. Protection of user credentials when accessing cloud environment

Proper security of the stored credentials (logins and passwords) is one of the most basic aspects of Melodic platform security. Due to the critical nature of these data, it is crucial to exercise due diligence in terms of their protection from disclosure and unauthorized access.

In order to limit distribution of the credentials among many platform components it has been established that they will be stored in one component only (Cloudiator) and encrypted symmetrically basing on AES algorithm with 256 bit key. Encryption/decryption key will be provided by Melodic

users during the first storing of credentials at the initial stage of application deployment before the deployment process starts and will be only known to these users.

Storage and security methods used to protect credentials on Melodic platform ensure sufficient security level.

It should be noted however that there is no possibility of re-entering (modifying) the credentials stored. It creates limitations in terms of password change policy, necessity of changing access account or in a situation where the user of the cloud environment resets his/her password.

3.2. Authentication of users and components

Due to the multi-module structure of the platform, its distributed architecture and the necessity of communication between many elements, it is crucial to provide uniform, proven and safe method of authentication for each operation taking place in the set of elements consisting of a user, a system component and an external system. The authentication should be based on username and password, but neither user's nor component's credentials should be spread between different elements of the platform. Such data should be stored in one place only and for authentication of each operation a security token of pre-determined and configurable validity period should be used.

Token-based authentication allows for resigning from spreading the username and password between different platform components, moreover, due to token's statelessness, it simplifies system scalability and makes the authentication mechanism independent from client's environment (different browsers, mobile devices etc.).

All above-mentioned requirements are met by the verified, commonly used and safe Security Assertion Markup Language 2 (SAML 2). SAML 2 is in fact a standard for securing the process of authentication in modern cloud applications. Using this standard in Melodic platform along with JWT (JSON Web Token) constitutes sufficient security of authentication of methods being called within particular components of the platform.

3.3. Authorization and access control

Authorization process on Melodic platform is executed in two different contexts:

1. Access control in terms of different resources (services, components, workflow and data).
2. Parameter validation (pre-authorization) of the application and datasets deployment plan before they are deployed on different cloud environments.

3.3.1. Access control

Melodic platform is a set of networked micro-services distributed around the Internet or virtual private networks. Distributed structure and necessity of communication between many components poses real threat of cyberattacks, which is why ensuring verified, safe, efficient and stable access control mechanism is crucial in the context of security of the whole platform.

Access control model selected by Melodic architects is the Attribute-Based Access Control (ABAC). Attribute-based access control uses pre-defined sets of policies and rules and verifies the set of different parameters, the so-called context (not only login and password for example) accordingly, to grant access to requested resources. The context might be:

- checking if access request is a part of regular workflow, or if it is only a standalone attempt,
- checking if the access attempt is performed in a correct sequence and time,
- requestor identity, his/her permissions, location, purpose of the operation,
- resource identity and its state,
- timeframes.

The method of ABAC model implementation in Melodic is XACML (eXtensible Access Control Markup Language) model and language based on Balana WSO2 library. XACML is a popular method of implementation of authorization services, commonly used in sectors such as: banking, healthcare or insurance.

Due to the use of multi-module architecture based on the following components:

- Policy Administration Point (PAP),

- Policy Enforcement Point (PEP),
- Policy Decision Point (PDP),
- Policy Information Point (PIP),
- Context Handler (CH),

XACML model guarantees adequate level of security, efficiency and stability for authorization services implementation based on this model.

It is important to stress that due to critical significance of authorization service in the context of functioning of the whole Melodic platform, it is crucial to ensure stability, redundancy and efficiency of the basic components of authorization server, which in the XACML model consists of:

- Policy Decision Point (PDP),
- Policy Information Point (PIP),
- Context Handler (CH).

ABAC-based authorization service in XACML implementation ensures proper security of Melodic platform resources, stability, efficiency as well as of the authorization service itself, meets all of the project requirements and is compliant with the best practices in this sector.

3.3.2. Pre-authorization

The idea of pre-authorization in the context of Melodic platform refers to validation of parameters of application and dataset deployment in cloud environments included in the deployment plan created by solvers. Elements that use pre-authorization are the Adapter and DLMS (Data Lifecycle Management System).

Pre-authorization means verifying compliance of the parameters included in the deployment plan such as:

- number and type of application components,
- way of dataset distribution between virtual machines (VM),
- selection of cloud provider,

- imposed requirements for virtual machines and datasets (including security requirements),
- selected installation procedures,

with adopted policies.

Pre-authorization allows deploying applications and datasets in compliance with existing laws, corporate standards or other restrictions adopted (i.e. budget) as well as protects from possible attacks on Upperware components, which might, in effect, lead to applying incorrect parameters in the application and dataset deployment plan.

3.4. Storing users' credentials

Due to specific permissions of Melodic users in the context of actions taken within maintained applications, it is important to ensure that users credentials stored on the platform are confidential.

Credentials should be stored in one central database providing data encryption, high efficiency, scalability and integration with third-party software.

Method used in the Melodic platform for storing users' credentials is central LDAP database based on OpenLDAP v.1.2 implementation.

The method adopted is compliant with the best practices in this sector. LDAP is an open-source protocol created for the purposes of authentication and authorization of users. It guarantees flexibility (due to the possibility of integration), scalability and security.

It is important to stress that only Melodic users' credentials are stored in the LDAP database. Application users' credentials are stored in respective application databases.

3.5. Users accountability

User activities within Melodic platform (modeling data and applications, determining requirements and objectives) directly affect stability, efficiency and scalability of the applications deployed by the

Melodic platform on cloud environments. That is why it is important to ensure full accountability of user activities within the platform. Such accountability should be based on the logs. They should include events that would allow determining who, where and when performed actions that might affect functionality and security.

It is worth considering introducing a differentiating mechanism for users access level depending on their needs or roles (i.e. read-only, full access, read-only log files, modeling application only, application deployment only etc).

3.6. Protection from unauthorized access to the user interface

Users' access to Melodic platform is executed via secure HTTPS protocol through the web browser, while authentication process uses credentials provided by the user (login and password). The credentials are stored in the internal LDAP database, basing on OpenLDAP v.1.2 implementation, while the communication between authentication and authorization mechanisms and the database is executed with the use of encrypted SSL connection.

While the choice of authentication, authorization and user credentials storage mechanisms is compliant with the best practices in this sector, their application alone does not provide sufficient security level in the context of protection from unauthorized access to the platform. Especially in case of possibility of accessing user interface (UI) via the Internet. Users' passwords are not covered by the complexity policy and multiple failed attempts of logging in do not result in locking the account. It makes the platform vulnerable to brute-force attacks, which might, in effect, lead to an unauthorized access to the user interface of the Melodic platform.

It is also worth noting that login attempts of users are not logged in the event logs.

In order to increase the level of security against unauthorized access, the following solutions should be considered:

1. Introduction of password complexity verification.

2. Introduction of account lockout mechanism after x failed login attempts.
3. Introduction of logging any login attempt, both successful and failed, in the event logs.

Due to the possibility of sharing user interface system via the Internet by the administrators or sharing Melodic platform as a PaaS service (Platform-as-a-Service) introduction of two-factor authentication mechanism should also be considered i.e. in the form of security tokens sent as a text message to user's phone number.

3.7. Securing REST communication between Melodic and cloud environments

Selected method of communication between Melodic platform and cloud environments is critical for security of applications maintained within the platform. Due to the critical meaning of data being sent between those environments, which, for instance:

- determine architecture of the application and its particular components;
- are control data, sent as a part of control plane;
- are metrics of particular components, sent as a part of monitoring plane;

ensuring confidentiality of these data is of the highest priority.

Communication between Melodic platform and cloud environments is executed with the use of the REST API. REST API as a web service is based on HTTP protocol, which, according to current trends, is a standard way of communication for distributed applications. Interaction between the client and the server is stateless, client's context is not stored on server between requests, it does not base on transactions and the way of communication between the client and the server is light and resource-oriented. One of the disadvantages of the REST method is that it lacks standardization (unlike i.e. SOAP). SOAP, thanks to standardization, provides many more possibilities, but at the same time it forces transmission of much more data between the client and the server, which makes it heavier and more complex than REST.

In order to ensure confidentiality of the transmitted data, each REST transaction should be authenticated and the communication between the server (cloud environment) and the client (Melodic platform) should be encrypted.

Using REST API and SSL-encrypted transmission for communication between Melodic platform and cloud environments is compliant with the best practices in this sector.

One should note however the existing risk of a Man-In-The-Middle (MITM) attack on encrypted HTTPS connections. The attacker redirects encrypted transmission to themselves and sends their own public key to both sides of the transaction, which, in effect, allows the attacker to intercept and modify transmitted information without their knowledge.

MITM attacks on API communications are rare and hard to execute. However, security mechanisms against such attacks provided by all cloud environments supported by the Melodic platform should be verified (security against MITM attacks is usually executed on the server side) and other mechanisms preventing sending of an unauthorized public key (certificate) for communication with cloud environments should be implemented.

3.8. High Availability

Criticality of the operations performed by the Melodic platform on applications and their data:

- constant monitoring of the application state,
- verifying the compliance of application and datasets deployment with current policy,
- controlling changes in application or datasets deployment

requires constant availability of the elements executing these operations.

Component integration method applied in the Melodic platform – ESB (Enterprise Service Bus) as well as its implementation based on MuleESB ensures statelessness of most of the key components, which enables running many instances of each of these components and, in effect, provides sufficient high availability mechanism. The only component that cannot work in a cluster

architecture (running multiple instances at the same time) is the stateful Learning Automate (LA) Solver. In case of a failure or an error LA Solver is restarted and starts counting all over again.

All of the above makes Melodic platform capable of constant support of the most critical applications.

4. RECOMMENDATIONS

All of the recommendations along with designation of their severity level are presented in table 1.

Severity level scale is as follows:

- high – meeting the requirement is critical in terms of platform security;
- medium – meeting the requirement is important in terms of platform security;
- low – meeting the requirement is optional in terms of platform security.

Table 1: Recommendations for amendments

No.	Recommendation	Severity
1.	Enabling modifications to cloud providers' credentials.	high
2.	Adding password complexity verification mechanism.	high
3.	Adding account lockout mechanism after x failed login attempts.	high
4.	Introducing accountability of user activity by logging all events that make it possible to determine who, where and when introduced modifications affecting functionality or security. Including all login attempts, both successful and failed.	medium
5.	Adding differentiating mechanism for levels of users' access to platform.	low

6.	Introducing two-factor authentication for Melodic users.	medium
7.	Guaranteeing confidentiality of data transmitted via REST communication between Melodic and cloud environments.	high
8.	Providing security against Man-In-The-Middle (MITM) attacks for communication between Melodic and cloud environments. Introducing public key (certificate) verification process.	medium

5. About the Authors

5.1. Slawomir Kobus

Co-founder of SIDIO and Managing Director responsible for the implementation and shaping of company's strategy in all areas of its activities.

Passionate about new technologies market with wide knowledge of mature manufacturers specializing in IT security and young, promising startups. As an experienced IT security consultant, he is particularly competent in projects implemented for financial institutions. During eighteen years of professional activity in IT companies, he went through all career levels, starting from the position of system engineer, through the positions of salesman and manager in international corporations. He implemented security systems for Telewizja Polska, Deutsche Bank, Bonduelle, Bank PKO BP.

Graduate of Electronics and Telecommunications at the Military University of Technology and postgraduate management at the Warsaw School of Economics. He completed Ethical Hacking training, holds certificates such as Cisco Certified Security Professional (CCSP), Checkpoint, Palo Alto Networks.

5.2. Adam Kuligowski

Co-founder of SIDIO responsible for determining the direction of company's development, cooperation with technological partners and technical implementation of projects.

He is a respected consultant and implementation practitioner, as well as a certified engineer for Fortinet, Palo Alto Networks, Proofpoint, Radware and Wallix. He collaborates with the largest Polish IT system integrators on advanced implementations for enterprises and corporations. At SIDIO, he supervises the technical aspects of Company's key projects.

A graduate of the Faculty of Cybernetics at the Military University of Technology in Warsaw and the Warsaw School of Economics in the field of "IT project management". He has been active in the IT security sector since 2011, first as an IT Security Engineer, responsible for numerous projects for clients from the public administration and commercial companies, and since 2016 as an Architect of IT Security Solutions at one of the largest IT integrators in Poland, carrying out projects for the most important public institutions, financial institutions and universities.

5.3. SIDIO

SIDIO is a team of practitioners and experts in the field of information and communication security. We are a group of engineers, implementers and consultants. Many years of practice, hundreds of implementations, consultations and trainings in the field of ICT network protection mechanisms have given us proficiency in selecting the best solutions, which very often have to be an effective compromise between data security and the comfort of users' work.

When creating SIDIO's offer, we have analyzed the criticality of ICT infrastructure elements and cases of cyber-attacks, both in terms of their diversity, most frequent vectors of implementation, as well as frequency of occurrence. From the perspective of data security and the entire IT infrastructure, we have considered the possibilities of mutual integration of protection systems we

have recommended. The result of this work is an offer based on the products of world technological leaders in the field of:

- email protection,
- network protection,
- endpoint and server protection,
- Web application protection,
- protection of intellectual property,
- raising awareness of IT security,
- privileged access management,
- security event monitoring and analysis,
- network access control,
- protection against DDoS attacks,
- incident response.