Deliverable D3.5

Tools and techniques for collecting evidence of technical and organisational measures – v2

Editor(s):	Anže Žitnik
Responsible Partner:	XLAB
Status-Version:	Final – v1.0
Date:	31.10.2022
Distribution level (CO, PU):	PU

Project Number:	952633
Project Title:	MEDINA

Title of Deliverable:	Tools and techniques for collecting evidence of technical and organisational measures – v2
Due Date of Delivery to the EC	31.10.2022

Workpackage responsible for the Deliverable:	WP3 – Tools to gather evidences for high-assurance cybersecurity certification
Editor(s):	Anže Žitnik (XLAB)
Contributor(s):	Immanuel Kunz, Florian Wendland (FhG), Franz Deimling (Fabasoft)
Reviewer(s):	Björn Fanta (Fabasoft), Olivia Kagerer (Fabasoft), Cristina Martinez (TECNALIA)
Approved by:	All Partners
Recommended/mandatory readers:	WP3, WP4, WP5, and WP6

Abstract:	This deliverable presents tools and techniques for the evidence collection of technical measures, such as security assessment of virtual machines, containers and server less functions or based on the analysis of information and data flows as well as organisational measures through the use of machine-learning and NLP. This is the second iteration of the tool integration, based on a refinement of the technical architecture after the first initial prototype was presented (D3.4). The third iteration will reflect the implementation of the use cases (D3.6). This deliverable is the result of Task 3.2, Task 3.3 and Task 3.4.
Keyword List:	Evidence gathering, Security assessment, Technical measures, Organisational measures, Components implementation, Clouditor, Codyze, Wazuh, Vulnerability Assessment Tools, Cloud Property Graph, Assessment and Management of Organisational Evidence
Licensing information:	This work is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/
Disclaimer	This document reflects only the author's views and neither Agency nor the Commission are responsible for any use that may be made of the information contained therein

(cc) BY-SA

Document Description

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	25.08.2022	First draft version - ToC	Anže Žitnik (XLAB)
	14.09.2022	Added contents to Sections 2, 3.2, 3.3	Anže Žitnik (XLAB)
	20.09.2022	Added contents to Sections 3.1, 4	Immanuel Kunz (FhG)
	23.09.2022	Updates to sections 3.2 and 3.3	Anže Žitnik (XLAB)
	27.09.2022	Added contents to Section 5	Franz Deimling (Fabasoft)
	29.09.2022	Minor updates in all sections, intermediate content review	Anže Žitnik (XLAB)
	07.10.2022	Updates in Sections 3.1 and 4	Immanuel Kunz, Florian Wendland (FhG)
	11.10.2022	Addressed comments in Section 3.1	Immanuel Kunz (FhG)
	12.10.2022	Addressed comments in Section 4	Florian Wendland (FhG)
v0.2	12.10.2022	Added executive summary, minor corrections across all sections, formatting, updates in Appendix A, comments	Anže Žitnik (XLAB)
	13.10.2022	Minor corrections in Section 4	Immanuel Kunz, Florian Wendland (FhG)
v0.3	13.10.2022	Formatting, updates in Appendix A, Conclusions. Ready for internal review.	Anže Žitnik (XLAB)
	21.10.2022	Internal review.	Björn Fanta, Olivia Kagerer (Fabasoft)
v0.4	31.10.2022	Addressing comments from the internal review.	Anže Žitnik (XLAB), Immanuel Kunz, Florian Wendland (FhG), Franz Deimling (Fabasoft)
v1.0	31.10.2022	Ready for submission	Cristina Martínez (TECNALIA)

CC BY-SA

Table of contents

(cc) BY-SA

(cc) BY-SA

List of tables

List of figures

CC BY-SA

Terms and abbreviations

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
AWS	Amazon Web Services
BSD	Berkeley Software Distribution
BSI	Bundesamt für Sicherheit in der Informationstechnik
CI/CD	Continuous Integration / Continuous Deployment
CLI	Command Line Interface
CloudPG	Cloud Property Graph
CPG	Code Property Graph
CPU	Central Processing Unit
CSA or EU CSA	EU Cybersecurity Act
CSP	Cloud Service Provider
CVE	Common Vulnerabilities and Exposures
DB	Data Base
DSL	Domain Specific Language
DLT	Distributed Ledger Technologies
EC	European Commission
ELK	ElasticSearch, Logstash, Kibana
EUCS	European Cybersecurity Certification Scheme for Cloud Services
GA	Grant Agreement to the project
GDPR	General Data Protection Regulation
GPL	General Public License
gRPC	Google Remote Procedure Call
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HIPAA	Health Insurance Portability and Accountability Act
HTTP	HyperText Markup Language
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
K8S	Kubernetes
KPI	Key Performance Indicator
LSP	Language Server Protocol
Nmap	Network Mapper
OASIS	Organization for the Advancement of Structured Information
OPA	Open Policy Agent
OS	Operating System
OWASP	Open Web Application Security Project
PaaS	Platform as a Service
PCI DSS	Payment Card Industry Data Security Standard
PoC	Proof of Concept
RAM	Random Access Memory
REST	Representational State Transfer
RPC	Remote Procedure Calls
SARIF	Static Analysis Results Interchange Format
SAST	Static Application Security Testing
SSL	Secure Sockets Layer

CC BY-SA

TLS	Transport Layer Security
UI	User Interface
URL	Uniform Resource Locator
YAML	Yet Another Markup Language
XML	Extensible Markup Language
VAT	Vulnerability Assessment Tools

(cc) BY-SA

Executive Summary

This deliverable presents the second (intermediate) version of the design, architecture, and implementation state of the evidence gathering components, being developed in the scope of Task 3.2, Task 3.3, and Task 3.4. It describes the components that produce evidence based on the assessment of cloud infrastructure (Clouditor, Wazuh, VAT), assessment of cloud applications source code (Cloud Property Graph and Codyze), and assessment of organisational measures (AMOE). It gives an overview of how these components relate and interact between each other and the rest of the MEDINA framework.

For each component, this document describes its purpose and scope, the (current and planned) coverage of MEDINA requirements, the component's internal architecture and its subcomponents, the external architecture and relation to other components, the implementation state at the point of producing this deliverable, and technical details of the component including the programming languages and frameworks used, information about the packaging and installation of the component, and licensing. It is also mentioned which EUCS [1] requirements the respective evidence gathering tool should cover.

This document is the successor of D3.4 [2], which presented the initial (in month 12) versions of the same mentioned components. D3.5 follows the same structure as well as keeps a part of content from the previous deliverable version to keep the document self-contained and easier to follow.

Other deliverables, closely related and worth reading for better understanding of the work presented herein, are D3.2 [3] and D5.2 [4], both released in parallel with D3.5. D3.2 is the result of Task 3.1 and Task 3.5, which deal with evidence gathering methodology, the integration of evidence gathering tools into MEDINA (Task 3.1) and maintaining the trustworthiness of evidence (T3.5). While this document contains the technical details about the implementation of evidence gathering tools, deliverable D3.2 explains the methodology behind the choice and design of evidence gathering components as well as an overview of the related state of the art. A further analysis of the EUCS requirements' coverage with all MEDINA tools is also presented in D3.2. The external architecture of components and the relationship with all other MEDINA tools is further described in the scope of the overall MEDINA architecture in D5.2, which also lists all MEDINA functional and technical requirements, elicited in WP5.

The presented components currently satisfy most of their functional requirements and are in a large part integrated with other components of the MEDINA framework. An overview of requirement fulfilment is presented in Appendix A.

The third and final version of this report will be delivered in D3.6 in month 30 (April 2023), presenting the final version of the above-mentioned components and their integration.

CC BY-SA

1 Introduction

Any automated compliance monitoring must start with the gathering of evidence upon which the analysis and decisions about the certification state can be made. The main technical contact point of MEDINA with the cloud services being evaluated are the evidence gathering components described in this deliverable.

1.1 About this deliverable

This is a report on the intermediate version of the design, implementation, and integration of the MEDINA evidence gathering components and is the second of three iterations (following D3.4 [2]) of the deliverable resulting from:

- Task 3.2, implementing the tools for assessing the security performance of cloud workloads and providing evidence about fulfilment of technical measures related to the operational cloud infrastructure,
- Task 3.3, implementing tools for assessing and collecting evidence about the security implications of cloud applications used and their data flows through analysis of the application source code,
- Task 3.4, implementing a component for the assessment of organisational measures based on the analysis of CSP's documentation about their policies and processes.

1.2 Document structure

This document is organised in the following sections:

1. *Introduction* (Section 1) gives the context for the results, reported in this document, its scope, structure, and mentions the relationship to other work in the MEDINA project as well as the modifications of this document in comparison with its first version, D3.4 [2].
2. *Evidence Management Tools High-level Architecture* (Section 2) gives an overview of the components, described in this document, and presents the architecture and relations between them.
3. *Security Assessment of Cloud Infrastructure* (Section 3) reports on the design and implementation of Clouditor, Wazuh, and Vulnerability Assessment Tools. The goal of these components is to provide evidence about conformity to technical measures regarding the cloud infrastructure and its configuration.
4. *Security Assessment of Cloud Applications* (Section 4) reports on the design and implementation of Cloud Property Graph and Codyze, components for cloud application source code analysis and provision of related technical evidence.
5. *Assessment of Organisational Measures* (Section 5) gives a report on the design and implementation of the component for extracting evidence of organisational measures from policy documents (AMOE).

Finally, Section 6 (*Conclusions*) summarizes and briefly comments on the reported results.

Appendix A gives an overview of the components' current implementation state.

Appendices B - E contain the user manuals and installation instructions of the individual described components.

(cc) BY-SA

1.3 Updates from D3.4

Please note that this document keeps some content that has been already included in D3.4 [2] and has not changed since then. Such material is kept in this deliverable to make it self-contained and easier to follow. For simpler tracking of progress and updates with regards to the previous deliverable version (D3.4), Table 1 shows a brief overview of the changes and additions to each of the document sections.

Table 1. Overview of deliverable updates with respect to D3.4

Section	Changes
2	Minor updates to the schema.
3.1	• Updated API descriptions, e.g., Clouditor certificate API • Updated requirements implementation status • Updated usage and installation information • Added features
3.2	• Updated descriptions of interfaces and connected components (newly implemented interfaces) • Updated status of the functional requirements • Extended technical description with minor previously unknown / unfinished details • Updated and extended the package information (for both Wazuh deploy and Wazuh & VAT Evidence Collector packages)
3.3	• Some details added regarding Wazuh & VAT Evidence Collector • Extended the description of the custom scripts' functionality • Updated status of the functional requirements • Minor updates to the technical description
4.1	• Updated description, including updates on published and planned papers • Updated functional description • Added section on relevant requirements and their implementation state • Added details on second iteration as advancements to MEDINA, especially regarding privacy analysis
4.2	• Extended description of Codyze and its components • Added introductory description of Codyze in MEDINA • Added remark for SARIF as format to report results to developers • Updated requirements to reflect progress • Updated architecture diagram better illustrating used components • Rewrote components description reflecting stronger differentiation between Codyze in MEDINA and existing public components • Updated section on advancements in MEDINA • Updated section on limitation and future work reflecting progress • Updated technical documentation, e.g., user manual and installation instructions
5	As the previous version only contained the preliminary component design, all implementation, requirements, and usage details are newly added.
Appendix A	Contents were updated according to the current implementation state of the components.
Appendix B	New section containing a README, user manual and installation instructions for Clouditor.

CC BY-SA

Appendix C	New section containing the user manual and installation instructions for Codyze.
Appendix D	New section containing the user manual and installation instructions for Cloud Property Graph.
Appendix E	New section containing the user manual for AMOE.

CC BY-SA

2 Evidence Management Tools High-level Architecture

This section gives a brief overview of the high-level architecture of MEDINA WP3 components, which result in the Evidence Management Tools. These components gather evidence about CSP's fulfilment of technical and organisational measures, perform initial processing of the evidence, and transmit it to other MEDINA components. The overall architecture of the MEDINA framework is further presented in D5.2 [4].

Figure 1 shows the architecture and data workflow among WP3 and other related components. Tools for collecting evidence about technical measures are represented at the bottom part of the figure. They are connected to the infrastructure under evaluation either through an interface of the underlying cloud provider or installed directly in the CSP's (virtual) machines. These components are further described in Section 3. Clouditor (Section 3.1) collects evidence about the secure configuration of cloud resources. Wazuh (Section 3.2) is installed in the CSP's cloud infrastructure and monitors the security state of the individual machines. Vulnerability Assessment Tools (Section 3.3) are also installed in the CSP's infrastructure and can periodically scan the configured servers and networks for vulnerabilities or run user-provided custom scripts for monitoring of specialized metrics and producing evidence based on the output.

Technical evidence, obtained from the analysis of cloud applications' source code is gathered by Codyze (Section 4.2). Cloud Property Graph (Section 4.1) can also gather evidence based on the analysed source code, but is not included in the architectural diagram since it is not yet integrated with the other components (see explanation in Section 4.1.1.1.1). Evidence about technical measures can also be collected by custom CSP-native components.

The component for organisational evidence gathering and processing (Section 5) analyses various documents and policies of the CSP and based on this produces evidence about the CSP's compliance to organisational requirements of the certification framework.

For further processing, evidence produced by all the mentioned components must be transformed into security assessment results with the information whether the addressed metric measured on the particular evaluation resource (e.g., virtual machine, cloud computing resource, storage, process, policy) is compliant or not compliant. The assessment results can be either produced by the evidence collection components internally and sent directly to the Orchestrator or by the specialized Security Assessment component.

The Security Assessment component assesses the received evidence based on the target values coming from the certification specification and CSP's configuration. For each evidence object, Security Assessment outputs a security assessment result with the information whether the addressed metric measured on the particular evaluation resource (e.g., virtual machine, cloud computing resource, storage, process, policy) is compliant or not compliant. The component is implemented as part of the Clouditor framework.

The assessment results and associated evidence are all gathered by the Orchestrator (also implemented as part of Clouditor). It stores these data in the respective databases and makes it available to the other components, mostly parts of WP4 and WP6 (e.g., Continuous Certification Evaluation, Company Compliance Dashboard). Evidence and assessment results are also forwarded to the Evidence trustworthiness Management system which uses Blockchain technologies to ensure the authenticity of data when it's retrieved at a later stage.

CC BY-SA

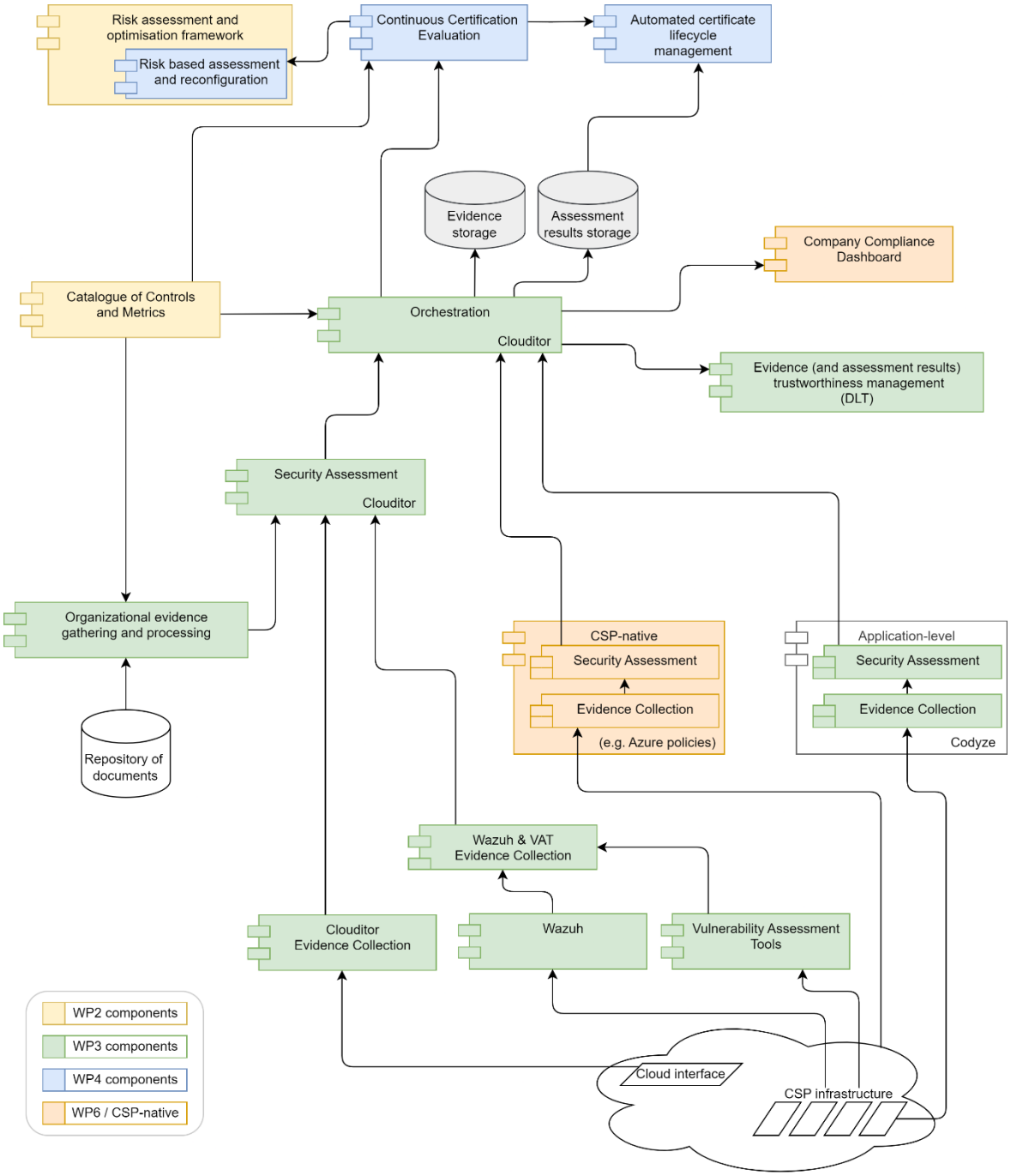

Figure 1. WP3 Architecture and directly related components (source: D3.2 [3])

CC BY-SA

3 Security Assessment of Cloud Infrastructure

This section describes the technical structure and implementation state of the components responsible for collecting evidence about the security performance of cloud workloads (cloud configuration, virtual machines and containers, or software running in them). The following subsections present the individual respective components, all being developed in the scope of Task 3.2.

3.1 Clouditor

Clouditor is a tool that comprises several components in the MEDINA framework, (i.e., the Cloud Evidence Collector, the Security Assessment, and the Orchestrator). Note also that in comparison to the previous iteration of this deliverable (D3.4 [2]), some names have been aligned.

Clouditor is a monitoring tool for cloud systems that can automatically and continuously discover existing resources in a cloud system, query their configuration, assess the gathered information according to pre-defined metrics, and more. These metrics may be mapped to certification frameworks, such as the EUCS [1], to demonstrate compliance with the certification requirements.

3.1.1 Implementation

3.1.1.1 Functional description

There are three components that make up Clouditor: The Evidence Collector which discovers cloud resources and creates MEDINA evidence for them, the Security Assessment which uses metrics to assess evidence and creates assessment results, and the Orchestrator which manages the evidence and assessment result flow, storage, and other utility functionalities.

The current implementation of Clouditor supports several cloud systems, i.e., Microsoft Azure, Amazon Web Services, and Kubernetes. The resource configurations in these platforms are checked by the use of various metrics. Examples of resource configuration checks are the following:

- Secure transport encryption with TLS
- Secure TLS version
- Data at rest encryption in various storage resources
- Resource deployment in allowed regions

Related requirements, common for all Clouditor's components

The relevant requirements from Deliverable D5.2 [4] are listed below with a brief description of how they are implemented. The requirements are grouped by components of Clouditor.

Requirement id	TEGT.C.01
Short title	Continuous collection
Description	The developed tools must be able to collect evidence continuously, i.e. in (high)-frequency intervals.
Implementation state	Fully implemented

(cc) BY-SA

Currently, the discovery interval in the Evidence Collector component is set to 5 minutes and can only be changed in the source code. In the future, this will become a customizable configuration through a configuration file or UI.

Requirement id	TEGT.C.02
Short title	Provision to defined interfaces
Description	The developed tools must provide collected evidence to a security assessment tool via its offered APIs.
Implementation state	Fully implemented

The Cloud Evidence Collector sends evidence to the Security Assessment component via its offered APIs.

Related requirements for the Evidence Collector component

Requirement id	TEGT.S.01
Short title	Collect evidence from cloud interfaces
Description	The developed tool must be able to collect evidence of cloud workloads, e.g., virtual machines, containers, and serverless functions.
Implementation state	Fully implemented

The Cloud Evidence Collector gathers evidence of cloud workloads from different CSPs (Azure, AWS, etc.). Resources are currently discovered in compute, storage, and network services in Azure, compute and storage services in AWS, and compute and network services in Kubernetes. These will be extended in the upcoming iteration.

Requirement id	TEGT.S.09
Short title	Collect evidence from CSP-native services
Description	The developed tool should be able to query findings from CSP-native services, like Azure Policy, to integrate them in MEDINA by querying the respective cloud API.
Implementation state	Not implemented

Currently, no CSP-native evidence collection is implemented (which is an optional requirement). This implementation would benefit a CSP by integrating security assessment results from existing security posture management systems, such as Microsoft Azure Security Center.

Related requirements for the Security Assessment component

Requirement id	EAT.01
Short title	Evidence assessment target
Description	The target values for the evidence assessment must be retrieved from a central repository of target values (WP2).
Implementation state	Fully implemented

The Security Assessment component retrieves target values from the Orchestrator, which in turn retrieves them from a central repository.

Requirement id	EAT.02
Short title	Continuous evidence assessment

(cc) BY-SA

Description	All evidence collection tools must forward evidence and measurement results (according to the data format defined in MEDINA) to the respective assessment components.
Implementation state	Fully implemented

The Cloud Evidence Collector sends evidence to the Security Assessment by using the provided APIs. They are then used to generate assessment results which indicate if the evidence is compliant or not.

Requirement id	EAT.03
Short title	Evidence assessment results
Description	The assessment results of evidence assessments must be submitted to the evidence Orchestrator via the API it provides.
Implementation state	Fully implemented

The Security Assessment component submits the assessment results by using the provided Orchestrator APIs.

Requirement id	EAT.04
Short title	Assess CSP-Native evidence
Description	The developed tool should be able to assess the CSP-native evidence or translate CSP-native assessment results to the MEDINA data model.
Implementation state	Not implemented

Currently, no CSP-native evidence collection is implemented (which is an optional requirement).

Related requirements for the Orchestrator component

Requirement id	ECO.01
Short title	Provision of Interfaces
Description	The evidence Orchestrator must provide standard interfaces for the evidence collection and assessment tools (T3.2-T3.4) to securely store their results.
Implementation state	Fully implemented

Interfaces are provided by RPC (Remote Procedure Call) APIs with gRPC[1], as well as via REST. Currently, the assessment tools send assessment results accompanied by the evidence they are based on. The transmission of evidence to the database can be encrypted.

Requirement id	ECO.02
Short title	Conformity to selected assurance level
Description	The evidence Orchestrator must ensure that the evidence collection (T3.2-T3.4) is performed according to the selected assurance level, i.e., it must trigger the evidence collection of the respective tools.
Implementation state	Partially implemented

[1] https://grpc.io/

CC BY-SA

Currently, the Cloud Evidence Collector is triggered via a CLI (Command Line Interface) command. Then the collected evidence is sent to the Security Assessment and the generated assessment results are then sent to the Orchestrator which stores them in a database. In the future, the selected assurance level will be addressed via the selected metrics that should be assessed.

Requirement id	ECO.03
Short title	Secure Transmission to evidence storage
Description	The evidence Orchestrator must securely transmit evidence to the evidence storage.
Implementation state	Fully implemented

The Orchestrator can store evidence either in memory or in a persistent database. In the latter case, the evidence can also be transmitted in an encrypted form, simply by specifying a SSL URL.

Requirement id	ECO.04
Short title	Transmission of evidence checksums
Description	The evidence Orchestrator should integrate a Ledger client that stores checksums of evidence in a DLT.
Implementation state	Fully implemented

The Orchestrator transforms assessment results into the desired format and forwards them to the ledger client.

3.1.1.1.1 Fitting into overall MEDINA Architecture

The three microservices that make up Clouditor constitute central components in the MEDINA framework as described in the following.

The **Cloud Evidence Collector** gathers evidence from cloud workloads. As such, it is one of the evidence collectors that can be integrated into the MEDINA framework.

The **Security Assessment** first retrieves metrics and target values from the Orchestrator and then assesses any incoming evidence accordingly. The Security Assessment can integrate with multiple evidence collectors. This way, CSPs can develop their own evidence collectors, and simply let them send evidence to the (Clouditor) Security Assessment.

The **Orchestrator** is the central management component of MEDINA which manages database access, cloud services, a user interface, and more. If CSPs decide to implement a custom Security Assessment, they can integrate it with the Orchestrator according to the MEDINA data model.

The Clouditor Security Assessment currently processes the evidence of the Cloud Evidence Collector, as well as evidence of other evidence collection tools, e.g., Wazuh. The Orchestrator processes the results of the Clouditor Security Assessment as well as results of other security assessment tools, e.g., Codyze.

Figure 2 shows an overview of the current Clouditor architecture.

CC BY-SA

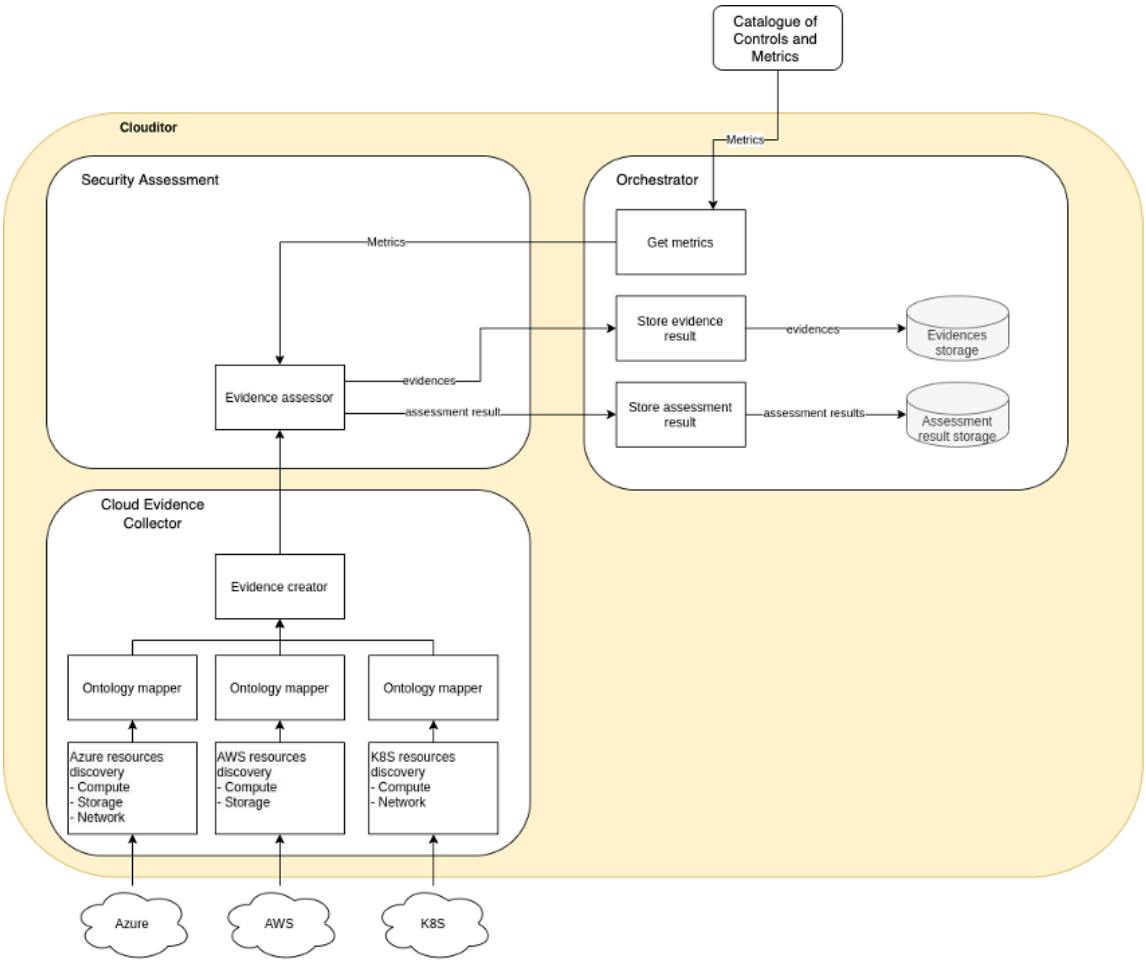

Figure 2. Overview of the Clouditor architecture

3.1.1.2 Technical description

In the following sections, we give the technical description of Clouditor's components. First, the architectural design is presented consisting of the architectural view and the connection between the respective components. Then, information about the single components is presented and, finally, an overview of the technical description for the implementation of the prototype is given.

3.1.1.2.1 Prototype architecture

Clouditor employs a microservice architecture allowing individual components to scale and to be replaced, or allowing to add new components, e.g., adding evidence collection tools for new cloud services/providers. The Evidence Collector, Security Assessment and Orchestrator are such modular components that represent microservices. Like all parts in Clouditor, they are written in Go and communicate among each other via the gRPC protocol.

Since the architecture is defined by the three components and the communication between them, interface snippets of the individual components are provided below. For the detailed specification see the *./proto* folder within the Clouditor repository[2]. The specification is defined in the *Protocol Buffer Version 3 Language Specification*. In addition, see the *./openapi* folder

[2] https://github.com/clouditor/clouditor

CC BY-SA

containing for each component the corresponding auto generated *.yaml* files which follow the OpenAPI description for REST APIs.

Cloud Evidence Collector interface

Table 2. Overview of the Cloud Evidence Collector's API functions

Function Name	Parameters	Return Type	Description
Start	-	successful (bool)	Triggers the start of the discovering process. Returns true if the component started without errors.
Query	filtered_type (string)	results (list of evidence)	Returns the latest set of evidence discovered.

Security Assessment interface

Table 3. Overview of the Security Assessment module's API functions

Function Name	Parameters	Return Type	Description
TriggerAssessment	options (string)	-	Triggers the security assessment.
ListAssessmentResults	-	results (list of assessment results)	Lists the latest set of assessment results.
AssessEvidence/ AssessEvidences	evidence (Evidence) / evidences (stream of Evidence)	successful (bool)/ -	Assesses the evidence/ stream of evidences provided by the evidences collection tool.

Orchestrator interface

Table 4. Overview of the Orchestrator's API functions

Function Name	Parameters	Return Type	Description
RegisterAssessmentTool	tools (AssessmentTool)	tool (AssessmentTool)	Registers the assessment tool
GetAssessmentTool	tool_id (string)	tool (AssessmentTool)	Returns the assessment tool with the given tool id.
UpdateAssessmentTool	tool_id (string), tool (AssessmentTool)	tool (AssessmentTool)	Updates the assessment tool given by the tool id.
DeregisterAssessmentTool	tool_id (string)	-	Deregisters the assessment tool with the given tool id.
StoreAssessmentResult/ StoreAssessmentResults	result (AssessmentResult) / results (stream of AssessmentResult)	-	Stores the assessment result/ stream of assessment results

CC BY-SA

Function Name	Parameters	Return Type	Description
			provided by the assessment tool.
StoreEvidenceResult/ StoreEvidenceResults	result (EvidenceResult)/ results (stream of EvidenceResult)	-	Stores the evidence provided by an assessment tool.
GetMetric	metric_id (string)	Metric	Returns the metric with the given metric id.
ListMetrics	-	List of Metrics	Returns a list of all metrics provided by the Catalogue of metrics and security schemes.
GetCertificate	Certificate ID	Certificate	Gets a certificate, e.g. an EUCS certificate
ListCertificate	-	List of certificates	Returns a list of certificates
CreateCertificate	Certificate	Certificate	Validates, stores, and returns the created certificate
UpdateCertificate	Certificate, Certifiate ID	Certificate	Validates, updates, and returns the updated certificate
RemoveCertificate	Certificate ID	-	Deletes the certificate
GetCloudService	Cloud Service ID	Cloud Service	Returns the Cloud Service
ListCloudServices	-	List of Cloud Services	Returns a list of Cloud Services
RegisterCloudService	Cloud Service	Cloud Service	Validates, stores, and returns the created certificate
UpdateCloudService	Cloud Service, Cloud Service ID	Cloud Service	Validates, updates, and returns the updated certificate
RemoveCloudService	Cloud Service ID	-	Deletes the Cloud Service
GetMetricImplementation	Metric Implementation	Metric Implementation	Gets a Metric Implementation
ListMetricImplementations	-	List of certificates	Returns a list of Metric Implementations
CreateMetricImplementation	Metric Implementation	Metric Implementation	Validates, stores, and returns the created Metric Implementation
UpdateMetricImplementation	Metric Implementation, Metric Implementation ID	Metric Implementation	Validates, updates, and returns the updated Metric Implementation

CC BY-SA

Function Name	Parameters	Return Type	Description
RemoveMetricImplementation	Metric Implementation ID	-	Deletes the Metric Implementation
GetCatalog	Catalog ID	Catalog	Gets a Catalog
ListCatalogs	-	List of Catalogs	Returns a list of Catalogs
CreateCatalog	Catalog	Catalog	Validates, stores, and returns the created Catalog
UpdateCatalog	Catalog, CatalogID	Catalog	Validates, updates, and returns the updated Catalog
RemoveCatalog	CatalogID	-	Deletes the Catalog
GetCategory	Category ID	Category	Gets a Category
ListCategory	-	List of Categories	Returns a list of Categories
GetControl	Control ID	Control	Gets the Control
ListControl	-	Controls	Returns a list of Controls
GetTargetOfEvaluation	Target Of Evaluation ID	Target Of Evaluation	Gets a Target Of Evaluation
ListTargetsOfEvaluation	-	List of Targets Of Evaluation	Returns a list of Targets Of Evaluation
CreateTargetOfEvaluation	Target Of Evaluation	Target Of Evaluation	Validates, stores, and returns the created Target Of Evaluation
UpdateTargetOfEvaluation	Target Of Evaluation, Target Of Evaluation ID	Target Of Evaluation	Validates, updates, and returns the updated Target Of Evaluation
RemoveTargetOfEvaluation	Target Of Evaluation ID	-	Deletes the Target Of Evaluation

3.1.1.2.2 Description of components

This section presents the tools provided by Clouditor, describing how they have been and will be further developed to meet the MEDINA requirements.

Evidence Collection

The functionality of the Evidence Collection can be divided into 3 parts:

- Fetching relevant properties of cloud resources (discovery),
- Creation of evidence objects, including their ontological concepts (see also D2.4 [5]), and
- Forwarding this evidence to the Security Assessment component.

Within the Cloud Evidence Collector, the discovery package is located at the top-level. Its purpose is to communicate with other services/components (in this case the Assessment component). In a first step, this service establishes a connection to the Assessment component, then it starts the various discoverers (e.g., for AWS S3), and forwards the collected evidence in a continuous manner. The transmission is done via a gRPC channel.

CC BY-SA

For each cloud vendor there is a separate sub package, e.g., for AWS and Azure. In such a package there is one file (e.g. *aws.go*) containing the cloud vendor-specific discoverer which loads and initializes configurations and credentials that all underlying services share. For each discovered cloud service, there is a corresponding Go file that fetches the desired properties of that service via API calls (programmatic access). According to the ontology defined in WP2, these properties are then converted into a format that is independent from the used cloud vendor. The properties that can be fetched are dependent on the range of API calls the respective cloud vendor provides.

For Microsoft Azure, the currently discoverable services are *compute, blob storage* and *network.* In the case of Amazon Web Services, *compute* as well as *blob storage.* Through the Kubernetes API *compute* and *network* resources are currently discoverable.

Security Assessment

Clouditor's Security Assessment is responsible for evaluating incoming evidence and sending the generated assessment results to the Orchestrator.

Evidence is received from components – the evidence collectors – such as the Cloud Evidence Collector or the Wazuh & VAT Evidence Collector. As mentioned above, Clouditor follows a microservice architecture which allows any evidence collector to connect to it in a modular way. Such evidence collection tools only need to implement the MEDINA API in gRPC to send evidence as Protocol Buffer messages. Additionally, REST over HTTP is available for evidence collecting tools. The gRPC approach, however, allows to send evidence in a stream which can significantly increase the throughput.

In a previous version of Clouditor, a dedicated policy rule language was used to assess evidence. Since no other tools outside the Clouditor tool suite needed to be connected to it, this approach was sufficient. In MEDINA, however, various evidence collection tools may connect to Clouditor – either to the Security Assessment or to the Orchestrator. To simplify the definition of policies, a more commonly used policy langue, Rego from Open Policy Agent (OPA)[3], was introduced instead. OPA uses Rego as a uniform declarative policy language. A policy written in Rego asserts that an input (e.g., an evidence) conforms to user-specified constraints (target values and operators).

In the definition of Rego policies, the cloud resource ontology (see D2.4 [5]) is used. Since evidence provided by the evidence collection tools provide their ontological assignments, the Rego policies only need to specify rules based on properties following the format of the ontology. Consider the following example: A policy checks if an encryption algorithm's key length is larger than the given target value, e.g., 256 bits (see Figure 3). The user-specific constraint may then state that the algorithm's key length must be at least 256 bit long (see Figure 4). The input, i.e., the evidence, is illustrated in Figure 5. The algorithm version in the input is 256, therefore the policy engine will output the compliance state of true. Both, input and the policy written in Rego, are aligned with the cloud resource ontology. These policies can be written more easily by non-experts without having to know how evidence collection, assessment, orchestration, etc. work. The person defining the policy only needs to know the ontology to write policies based on it.

[3] https://www.openpolicyagent.org/docs/latest/policy-language/

CC BY-SA

In the MEDINA framework, the Rego policies are generated from metrics which are stored in the Catalogue of controls and metrics component. When the Orchestrator triggers the assessment to start, it also sends the respective metrics along. The assessment then stores these metrics in cache for fast processing of the evidence.

The outcome of these assessments, the assessment results, are then sent to the Orchestrator and will eventually reach the continuous certification evaluation component (see D4.2 [6]).

```
default compliant = false

compliant {
    data.operator == ">="
    input.atRestEncryption.algorithm >= data.target_value
}

compliant {
    data.operator == "=="
    input.atRestEncryption.algorithm == data.target_value
}
```

Figure 3. Sample policies written in Rego: They compare a given encryption algorithm to a given target value (see next figures), depending on a given operator

```
{
    "operator": ">=",
    "target_value": 256
}
```

Figure 4. Sample data that will be provided in the future by the central catalogue of metrics and target values

```
{
    "atRestEncryption": {
        "algorithm": 256,
        "enabled": true,
        "keyManager": "Microsoft.Storage"
    },
    "name": "storage12"
}
```

Figure 5. A sample excerpt of an evidence

Orchestrator

The Orchestrator is a central component in the MEDINA framework and acts as a central management component for launching WP3 components and orchestration of dataflows between components. As such, it also manages the interaction between components of different work packages. The Orchestrator also offers APIs to store and retrieve data and manage assessment tools. The APIs are defined in gRPC allowing other components to only implement the given API to send the data as Protocol Buffer messages. For some APIs it is also possible to send the data in a stream which can increase the throughput. Its interactions with other components and its functionalities are summarized in the following:

CC BY-SA

- The Orchestrator exposes two APIs for the security assessment tools, e.g., Clouditor Security Assessment, CSP-native or Codyze security assessment tool. One API is for the assessment results and another one to store evidence directly.
- The Orchestrator also acts as the central interface to the Catalogue of controls and metrics which is developed within WP2 (see deliverable D2.1 [7]). As such, it is responsible for providing relevant metrics to the assessment component.
- Furthermore, The Orchestrator stores checksums of evidence and assessment results in the DLT via a Blockchain client. The implementation of the Trustworthiness Management System component is also described in D3.2 [3].
- Additionally, The Orchestrator forwards assessment results to the Continuous Certification Evaluation component which is developed within WP4 (see deliverable D4.2 [6]).
- The Orchestrator stores the evidence as well as the assessment results into the associated storages. It offers an in-memory storage as well as a PostgreSQL connection.

3.1.1.2.3 Technical specifications

The prototype is written in Go (version 1.19). A selection of key libraries is shown in the following and a full list of used libraries can be found in the Github repository[2].

- *github.com/Azure/azure-sdk-for-go*
- *github.com/aws/aws-sdk-go-v2*
- *k8s.io/client-go*
- *google.golang.org/grpc*
- *google.golang.org/protobuf*
- *gorm.io/driver/postgres*
- *gorm.io/driver/sqlite*

Either an in-memory DB or a postgres DB can be used.

3.1.2 Delivery and usage

The following sections give a short overview of the delivery and usage of the prototype. Further technical details can be found in the Clouditor Github Repository[2].

Please note that the README, installation instructions and user manual of Clouditor can be found in Appendix B.

3.1.2.1 Package information

Table 5 shows the structure of the important folders and a brief description.

Table 5. Overview of the Clouditor package structure

Folder	Description
api/	Code needed for the communication between the microservices. It mainly consists of auto-generated *Protobuf* and gRPC files.
cli/	This folder contains the Clouditor CLI based source code files.
cmd/	This folder contains the main files.
openapi/	This folder contains the auto-generated *OpenAPI* files.
persistence/	This folder contains the DB specific files.
policies/	This folder contains the *Rego* policy files per metric.
proto/	This folder contains the *Protobuf* files.

CC BY-SA

rest/	This folder contains the REST gateway implementation.
service/	This folder contains the source code for the microservices separated in individual folders for each service.
voc/	This folder contains the vocabulary files based on the ontology defined in WP2.

3.1.2.2 Licensing information

The Clouditor components are licensed under the open-source Apache Licence 2.0.

3.1.2.3 Download

The Clouditor source code can be found in the Clouditor Github repository[4]. The adapted MEDINA components can be found in the public MEDINA repository[5].

3.1.3 Advancements within MEDINA

Several modifications and features have been implemented in Clouditor within the first and second year of the MEDINA project:

- The component has been completely reimplemented in the Go programming language.
- The previous Clouditor architecture has been redesigned to create several microservices, e.g., separate microservices for evidence gathering, assessment, and orchestration. This modularization allows for better scalability, as well as allows to integrate alternative services, for instance other evidence gathering tools.
- The evidence gathering service has been extended with an ontology mapping, i.e., the resource properties that are discovered are enhanced with a mapping to a cloud resource ontology. For example, a virtual machine's properties are extended with a mapping to the ontology concepts *computing* and *virtual machine*. This approach allows to define metrics independently from the cloud provider and certification catalogue. For information, please refer to the respective description in deliverable D2.4 [5] (cloud resource ontology).
- As described above, the assessment service has been reimplemented as a separate microservice as well to conform to the MEDINA guidelines and data model. Also, its usage of the OPA[6] policy engine has been added, which is used to evaluate incoming evidence against metrics and their target values. These are defined using the OPA policy language Rego.
- The Orchestrator service is a completely new component in Clouditor, i.e., its APIs, data model, and integration with other components has been designed and implemented from scratch within MEDINA.
- The three components have been integrated and tested with each other, as well as with other MEDINA components, such as the Continuous Certification Evaluation and the Blockchain client.
- Additional APIs have been implemented, for instance for the update of metric target values.
- Integration of the various components with the central OAuth server.
- Improvement of the data model and persistence.

[4] https://github.com/clouditor/clouditor
[5] https://git.code.tecnalia.com/medina/public
[6] https://www.openpolicyagent.org/

CC BY-SA

- Introduction of data entities and APIs for adding cloud services and certification frameworks.
- Introduction of the Target of Evaluation which binds a cloud service to a certification framework and supports an n:m relation between cloud services and certification frameworks to be evaluated.

3.1.4 Limitations and future work

Cloud Evidence Collector

The Cloud Evidence Collector, which currently collects evidence from Microsoft Azure, AWS, and Kubernetes systems, is limited by the access rights it is given in the respective user management system, such as Azure Active Directory. Therefore, it will only measure the resources that are visible to its given user. Furthermore, cloud provider APIs may change, so the component needs to be updated accordingly. If, for instance, relevant security properties like access control properties change, their inclusion in the MEDINA evidence needs to be aligned in this component. Also, the evidence collection is limited by the information that the cloud provider APIs implement: if a certain encryption property, for example, would not be implemented by an API, the evidence collection for that property would not be possible. Since Cloud Evidence Collector adds ontological terms to the evidence, also limitations of the ontology need to be considered. First, the ontology terms need to be added correctly to the evidence or the Security Assessment will apply the wrong metrics to it. Second, the ontology needs to be maintained and its changes need to be implemented accordingly in the evidence collection.

Security Assessment

The Security Assessment component uses the Open Policy Agent (OPA) and Rego to perform the assessment of evidence against expected values (defined in the MEDINA metrics). OPA is in version 0.45 as of October 2022; future breaking changes therefore may occur which have to be incorporated in this component. It is furthermore dependent on the availability of the Orchestrator, since it must forward assessment results to the Orchestrator and receive metric implementations (Rego code) from it.

The Security Assessment also currently only applies a small set of metrics which we will expand according to the KPIs.

Orchestrator

The Orchestrator is the central management component in MEDINA. While it presents an efficient component for forwarding data, managing database accesses, etc., it is also a single point of failure for the framework since without it, no evidence or assessment results can be processed or stored.

In the upcoming iteration, new features of the Orchestrator will be integrated with the other MEDINA components, e.g., the support for multiple cloud services and the Target of Evaluation entity will be integrated with the Continuous Certification Evaluation.

3.2 Wazuh

Wazuh [8] is an open-source security monitoring tool for threat detection, integrity monitoring, incident response and basic compliance monitoring. It can be deployed on-premises or in hybrid and cloud environments.

(cc) BY-SA

3.2.1 Implementation

3.2.1.1 Functional description

Wazuh's role in MEDINA is to provide capabilities for threat detection to MEDINA users (CSPs) while producing evidence related to its usage and potentially detected threats. Wazuh's connection to MEDINA is enabled by the Wazuh & VAT Evidence Collector component. It connects to Wazuh to query its configuration and detected events and produces evidence based on this data.

Unlike some other evidence gathering tools (e.g., Clouditor), Wazuh is not primarily connected to the cloud interfaces, but its agents are installed directly on the (virtual) machines of the monitored infrastructure. The agents can run on many different platforms, such as Windows, Linux, Mac OS X, AIX, Solaris, and HP-UX. Wazuh includes several modules that each support their respective detection capability. For each of the modules, specific rules are defined that include internal metrics and thresholds to trigger events or alerts. When an alert is produced based on some detected event(s), additional actions can be triggered to notify a user or another component about it. With this capability, certain events (e.g., malware detected, Wazuh agent shutdown…), can trigger changes of values for specific MEDINA metrics and event-driven generation of evidence.

Wazuh's detection modules include the following:

- Occurrence of changes within system files (file integrity checks): Wazuh agent monitors the file system to detect changes in system files' content or attributes. Changes of system settings or other critical files can signify that the monitored machine is compromised.
- Detection of malware and rootkits installed on the infrastructure: Wazuh can scan the monitored system for various types of malware. It combines a signature-based approach for detecting suspicious programs with anomaly detection capabilities, detecting intrusions by monitoring system call responses. Signature-based malware detection is supported through integration with the open-source antivirus engine ClamAV [9] or VirusTotal [10], an online API for analysis of suspicious files.
- Number and severity of infrastructure vulnerabilities detected (e.g., CVE level of dependencies installed on the OS being monitored): Wazuh identifies the software installed on the monitored system and compares the versions with its online inventory in order to find software known to contain vulnerabilities.
- Monitoring cloud logs via IaaS or PaaS API: Wazuh includes modules for integration with some cloud providers' APIs (Amazon AWS, Azure, Google Cloud) to analyse security configuration of the cloud and notify about detected weaknesses.
- Compliance level with standards such as PCI DSS, HIPAA, GDPR: Wazuh integrates verification for some of the basic requirements of the mentioned standards. The Wazuh UI provides a dashboard with an overview of these requirements' fulfilment.

The main innovation of the usage of Wazuh and the extensions provided by MEDINA mainly lies in the flexibility of the proposed architecture. MEDINA can offer Wazuh and its extensions to the CSPs as a tool for incident detection and continuous monitoring of security indicators. Using Wazuh, compliance with several security controls can be automatically verified and the produced evidence integrated with MEDINA. The controls that can be satisfied with Wazuh relate to malware protection, logging, threat analytics, and automatic monitoring (alerting). The initial analysis of EUCS requirements covered by Wazuh is further described in D3.2 [3]. Beside

CC BY-SA

the provided functionalities, Wazuh also offers a platform for implementing custom detectors on the monitored machines and easily integrating them with MEDINA.

An example of collecting evidence with Wazuh is provided here for verifying the fulfilment of (draft) EUCS requirement **OPS-05.3H**. The requirement reads: "*The CSP shall automatically monitor the systems covered by the malware protection and the configuration of the corresponding mechanisms to guarantee fulfilment of above requirements, and the antimalware scans to track detected malware or irregularities.*" (the above requirements referring to the other requirements from the OPS-05 control: Protection against malware – implementation). According to the descriptions of these requirements, the conditions for regarding a machine compliant with OPS-05.3H as verified by Wazuh, are:

- Enabled file integrity monitoring module
- Enabled malware and rootkit detection module
- Enabled integration with ClamAV or VirusTotal for additional malware protection
- At least one alerting service enabled in Wazuh to automatically notify the responsible persons in case of detected alerts

The first three conditions ensure that malware protection is enabled, while the last condition verifies that automatic monitoring is configured as well. To verify all conditions, the Wazuh & VAT Evidence Collector component makes several API queries to Wazuh for each of the (virtual) machines in scope. An evidence object is produced for each of the monitored machines with a measurement value according to the obtained result – positive if (and only if) all the mentioned conditions are satisfied.

Related requirements

Below is the collection of requirements (described in D5.2 [4]) related to the component and a description of how and to what extent these requirements are implemented at this point of development.

Requirement id	TEGT.C.01
Short title	Continuous collection
Description	The developed tools must be able to collect evidence continuously, i.e., in high-frequency intervals.
Implementation state	Partially implemented

Continuous collection is implemented, but evidence is collected only for a limited number of metrics.

Requirement id	TEGT.C.02
Short title	Provision to defined interfaces
Description	The developed tools must provide collected evidence to a security assessment tool via its offered APIs.
Implementation state	Fully implemented

Interface between the Wazuh & VAT Evidence Collector and Clouditor (providing the security assessment capabilities) is implemented.

Requirement id	TEGT.S.08
Short title	Provision of malware and vulnerability detection tools

(cc) BY-SA

Description	Tools for malware detection, intrusion detection, and vulnerability scanning must be provided to assist CSPs with satisfying related requirements of security standards or to verify the compliance with such requirements.
Implementation state	Fully implemented

Wazuh offers capability of malware scanning and vulnerability detection of the infrastructure and applications (in some cases). Wazuh agents pull software inventory data and send this information to the Wazuh Manager, where it is correlated with continuously updated CVE databases, in order to identify well-known vulnerable software. Automated vulnerability assessment helps the user identify the weak spots of their critical assets. Integration through the Wazuh & VAT Evidence Collector allows MEDINA to verify the malware detection state and gather evidence about it.

3.2.1.1.1 Fitting into overall MEDINA Architecture

Wazuh is integrated with the rest of the MEDINA framework through the Wazuh & VAT Evidence Collector component that gathers evidence from both Wazuh and VAT. Wazuh is installed inside the CSP's infrastructure and gathers information about possible security threats of the system. The state of Wazuh's operation and the detected security events, gathered by Wazuh, are queried by the Wazuh & VAT Evidence Collector, which forwards such information to Clouditor's Security Assessment in the form of evidence. Figure 1 shows the positioning of Wazuh in the MEDINA architecture among the WP3-related components.

3.2.1.2 Technical description

The following subsections describe the technical details of Wazuh.

3.2.1.2.1 Prototype architecture

Wazuh is composed of a Wazuh server and multiple Wazuh agents. The agents are deployed on the individual monitored machines and communicate information about the detected anomalies to the server. In a cloud environment, the agents are deployed on the virtual machines inside the monitored cloud infrastructure, independent of the cloud provider. Wazuh server should be installed on a dedicated (virtual) machine, ideally in the same network segment as the agents or otherwise made available by the network routing rules.

The server includes the Wazuh manager component along with the ELK (ElasticSearch, Logstash, Kibana) stack for gathering, storing, and displaying data. Custom integrations are possible to send alerts from Wazuh to any external component.

The basic architecture of Wazuh is depicted in Figure 6. Looking at it from high-level, it consists of Wazuh Agents and Wazuh Server. The Wazuh agent (installed on endpoints) with different interfaces (modules) is able to detect different metrics on the host. Wazuh Server consists of worker nodes (Wazuh cluster), a Kibana Server that provides a web user interface for overview of all logs and relevant events, and an ElasticSearch database server that stores the logs and detected events, coming from the agents.

(cc) BY-SA

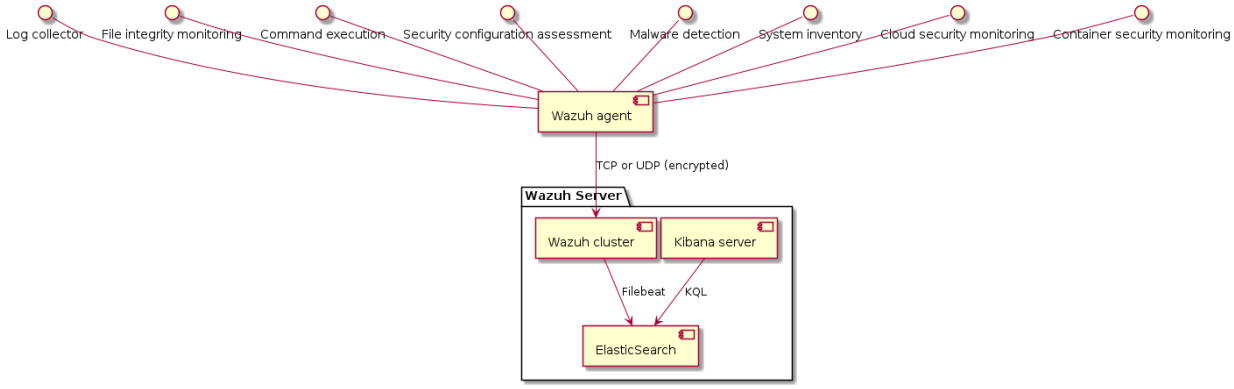

Figure 6. High-level Wazuh's architecture

3.2.1.2.2 Description of components

Wazuh Agents communicate with the Wazuh server using Rsyslog. Wazuh is plugged into MEDINA with the Wazuh & VAT Evidence collector component, which is responsible for extracting the data, relevant for MEDINA metrics, and transforming it into evidence compatible with the Security Assessment component. The Evidence collector communicates with the Wazuh server using HTTP (API exposed by Wazuh). It also includes a two-way communication with the Security Assessment component (Clouditor). This is depicted below in Figure 7.

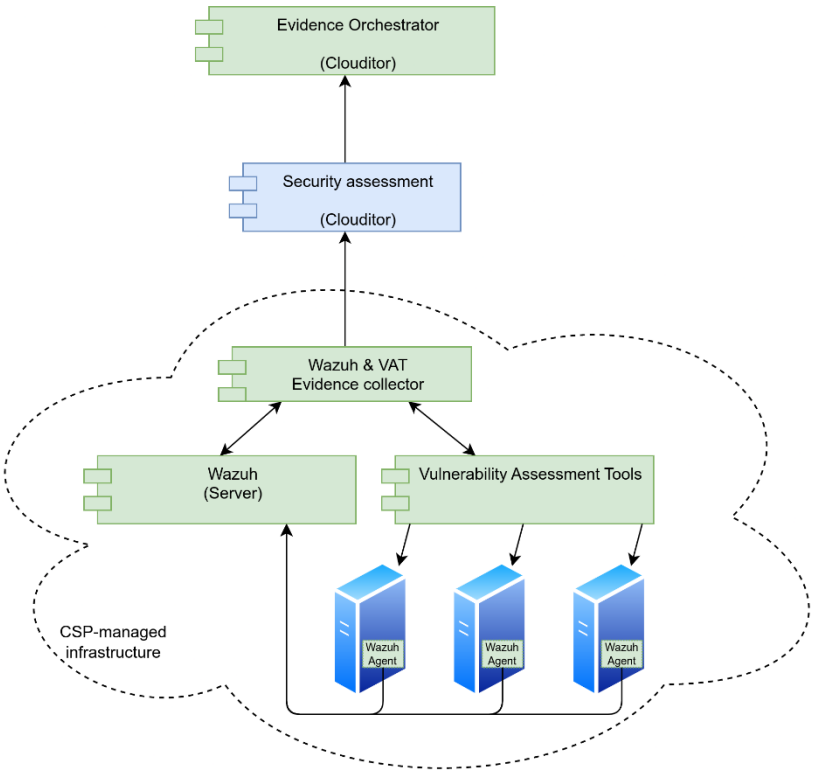

Figure 7. High-level schema of Wazuh, VAT, and related components

3.2.1.2.3 Technical specifications

The prototype's implementation consists of the following:

CC BY-SA

- MEDINA-specific deployment and configuration scripts for Wazuh (Ansible deployment scripts, YAML definitions, configuration). This also contains specific MEDINA configurations of Wazuh rules (XML, JSON).
- Wazuh & VAT Evidence Collector, a component that integrates Wazuh with the MEDINA Security Assessment. This component is developed in Python and packaged as a Docker container.

3.2.2 Delivery and usage

3.2.2.1 Package information

Wazuh deployment package

The Wazuh deployment package contains all the needed deployment and configuration scripts for installing Wazuh, Wazuh & VAT Evidence Collector, and also Clouditor (needed by the Evidence Collector to connect with). For demonstrative purposes and replicating a deployment on CSP's infrastructure, the process locally creates five virtual machines (using Vagrant): a Wazuh server, two Wazuh agents, Wazuh & VAT Evidence Collector, and Clouditor Security Assessment. Table 6 describes the important folders and files of this package.

Table 6. Overview of the Wazuh-deploy package structure

File / folder	Description
ansible/	This folder contains Ansible playbooks – scripts for installation of individual sub-components.
environments/	This folder contains several sets of configurations for different installation environments (based on the purpose, some components might not be installed or have various configuration applied – README file contains details).
custom-provision/	This folder contains the configuration set for installation on existing machines in the CSP's infrastructure.
Makefile	Contains simplified make scripts that trigger the installation procedures.
README.md	Contains details about installation requirements and instructions.

Wazuh & VAT Evidence Collector

The Wazuh & VAT Evidence Collector repository contains the source code of the connector between Wazuh (or VAT) and the MEDINA Security Assessment component (Clouditor). The source code is written in Python and contains a Docker file so that it can be simply built and used as a Docker container. Table 7 describes the important files and folders of this package.

Table 7. Overview of the Wazuh & VAT Evidence Collector package structure

File / folder	Description
forward_evidence/	This folder contains code responsible for packaging and sending evidence objects to Clouditor (Security Assessment).
grpc_gen/	This folder contains code, automatically generated based on the protocol buffer definitions of the Clouditor Security Assessment gRPC interface.
kubernetes/	This folder contains Kubernetes definition files for automated deployment on the MEDINA Kubernetes dev & test clusters.
proto/	This folder contains the protocol buffer definitions of the Clouditor Security Assessment gRPC interface.

CC BY-SA

scheduler/	This folder contains code responsible for scheduling of the evidence gathering process.
test/	This folder contains code for (self-) testing of the component.
wazuh_evidence_collector/	This folder contains the core code responsible for the communication with Wazuh API and building (internal) evidence objects.
Dockerfile	This file contains the definition for building the Docker image of Wazuh & VAT Evidence Collector.
resource_id_map.json	This file contains a human-readable and editable configuration – mapping of MEDINA resource IDs to target machines' hostnames.
README.md	Contains installation and configuration instructions.

3.2.2.2 Installation instructions

Requirements:

- Vagrant 2.2.14
- Ansible 2.9.16

Clone the Wazuh deployment repository (see 3.2.2.5 below). To setup the demo, simply provision the Wazuh server, Wazuh agents, Clouditor, and Evidence Collector virtual machines by running:

```
make create provision
```

For other installation options, please consult the README file in the repository.

3.2.2.3 User Manual

After installation on a local Vagrant environment, the Wazuh UI can be accessed by navigating a web browser to https://192.168.33.10:5601 (if using the default deployment configuration). Default credentials (admin:changeme) can be used for logging in the web interface.

After accessing the "Wazuh" section in the web UI, the user can notice two agents registered and running with Wazuh. Evidence Collector is configured to collect evidence about the malware detection running on the agent machines every minute. This can be inspected by examining the logs of the Evidence Collector virtual machine.

3.2.2.4 Licensing information

The core Wazuh [8] component is open source, licensed with a modified GPLv2 license[7].

The deployment scripts for the MEDINA proof-of-concept and the Wazuh & VAT Evidence Collector, developed by XLAB, are licenced with the open-source Apache Licence 2.0.

3.2.2.5 Download

The code of open-source components built by MEDINA is available on the project's git repository, on GitLab hosted by TECNALIA:

- Wazuh & VAT Evidence Collector:
  https://git.code.tecnalia.com/medina/public/wazuh-vat-evidence-collector

[7] https://github.com/wazuh/wazuh/blob/master/LICENSE

CC BY-SA

- Wazuh deployment repository:
  https://git.code.tecnalia.com/medina/public/wazuh-deploy

3.2.3 Advancements within MEDINA

Wazuh is a software solution developed independently of MEDINA by its respective owner, Wazuh Inc. In the scope of MEDINA, several advancements have been made in terms of integration with MEDINA, associated configurations and implementation of the Wazuh & VAT Evidence Collector component. The (non-exclusive) list of work done in the project is as follows:

- Analysis of the EUCS requirements to determine which of the requirements can be verified or satisfied by using Wazuh.
- Definition of architecture for collecting evidence with Wazuh and integrating it with MEDINA (Wazuh & VAT Evidence Collector).
- Implementation of Wazuh & VAT Evidence Collector with connection to Clouditor Security Assessment service.
- Implementation of CI/CD pipelines for automatic installation on the MEDINA "dev" and "test" deployments.
- Production of deployment scripts to enable easier installation of software on the pilots' infrastructure.
- Improvements of Wazuh & VAT Evidence Collector to support multiple metrics.
- Implementation of changes related to advancements in the MEDINA data model.

3.2.4 Limitations and future work

Evidence gathering with Wazuh is currently possible for a limited number of metrics related to the (draft) EUCS requirements OPS-05.3H, OPS-12.2H and OPS-21.1H. This limitation will be progressively removed by adding support for other metrics.

Wazuh uses various techniques for evidence gathering. By using the integrated anti-malware and intrusion detection systems, a CSP is satisfying the standardisation requirements. In this case, evidence is produced bearing the information about the functioning of Wazuh and its modules. Such evidence has a high level of confidence. If the CSP uses other (unrelated) tools for malware detection, the limitation is that an integration layer needs to be developed between those tools and Wazuh. While Wazuh's log collection capabilities make such integration relatively easy with most tools, support by the other tool might be limited.

Custom Wazuh rules can also be written to evaluate logs, coming from other services and produce events or alerts based on their contents. Evidence can in turn be produced based on such events or alerts. The level of confidence obtained in this way is fully dependent on the implementation of the Wazuh particular rule.

3.3 Vulnerability Assessment Tools

Vulnerability Assessment Tools (VAT) act as a vulnerability scanning and detection framework. VAT is intended to be deployed in the CSP's infrastructure and configured to periodically scan the machines and servers on the monitored network, using several tools to detect vulnerabilities.

(cc) BY-SA

3.3.1 Implementation

3.3.1.1 Functional description

VAT's collection of vulnerability scanning tools comprise two web vulnerability scanners (W3af [11] and OWASP ZAP [12]), a network discovery and auditing tool Nmap [13], and a framework for including user-defined custom scripts for detecting specific issues or simply notifying about unavailability of particular services.

Beside the vulnerability scanners, VAT is composed of several components supporting the scheduling of scanning tasks, definition of custom scripts for scanning or monitoring, as well as communication and integration with other MEDINA tools.

The innovation that VAT brings to MEDINA is the usage of vulnerability scanners for automated verification of compliance. There are several requirements of security standards that can be either satisfied with VAT or evidence can be gathered about their fulfilment. EUCS requirements covered by VAT include several from the vulnerability detection and management categories, usage of encrypted communication protocols, separation of networks and monitoring new devices on the network, etc.

Additional innovation is the modularity and flexibility of the VAT framework. Beside the included vulnerability detection tools, users can define their own scripts written in one of the several supported programming languages, or even integrate their own vulnerability scanning tools, depending on their specific needs. VAT's feature of including custom scripts for monitoring of metrics enables the user to easily provide their own code for checking of specific metrics. The output of such code (script) is automatically transformed into evidence and integrated into the MEDINA workflow. Beside the detection of vulnerabilities with the included tools and provision of related evidence, VAT framework thus also enables the implementation of custom detectors to produce other evidence types or monitor other CSP-specific networking metrics.

The coverage of EUCS requirements by VAT is also described in deliverable D3.2 [3].

Related requirements

Below is the collection of requirements (from D5.2 [4]) related to the component and a description of how and to what extent these requirements are implemented at this point of development.

Requirement id	TEGT.C.01
Short title	Continuous collection
Description	The developed tools must be able to collect evidence continuously, i.e. in (high)-frequency intervals.
Implementation state	Partially implemented

VAT framework enables configuration of the scanning tasks and continuous scanning with adjustable intervals and manually configured metrics. No evidence is currently collected from the results of the generic (integrated) vulnerability scanners.

Requirement id	TEGT.C.02
Short title	Provision to defined interfaces
Description	The developed tools must provide collected evidence to a security assessment tool via its offered APIs.

CC BY-SA

Implementation state	Fully implemented

The interface for communication between the component core API, the VAT & Wazuh Evidence Collector, and Clouditor (Security Assessment) is implemented.

Requirement id	TEGT.S.08
Short title	Provision of malware, intrusion, and vulnerability detection tools
Description	Tools for malware detection, intrusion detection, and vulnerability scanning must be provided to assist CSPs with satisfying related requirements of security standards or to verify the compliance with such requirements.
Implementation state	Fully implemented

VAT includes several vulnerability scanners and a framework for their orchestration and automated running of the scans. The possibility to add custom vulnerability scanning scripts is also implemented.

3.3.1.1.1 Fitting into overall MEDINA Architecture

The position of Vulnerability Assessment Tools inside the MEDINA architecture is depicted in Figure 1 (Section 2) and in slightly more detail in Figure 7. VAT scans the monitored machines inside the CSP's infrastructure, which is communicated to the Wazuh & VAT Evidence Collector component that constructs the evidence about fulfilment of the monitored metrics and sends them to the Security assessment component (Clouditor) for further processing.

3.3.1.2 Technical description

The following subsections describe the technical details of Vulnerability Assessment Tools.

3.3.1.2.1 Prototype architecture

The internal architecture of the Vulnerability Assessment Tools consists of several microservices (see Figure 8). The main components are: Scan Configurator (web user interface), Vulnerability Scanning Registry, Catalogue of custom scripts, and VAT Service Orchestrator with several subcomponents. The figure also shows an example of a user's request to issue a scan originating in web interface and the data flow through the other VAT subcomponents. The connection to other MEDINA components for evidence gathering is issued through the Wazuh & VAT Evidence Collection component (see also Figure 7), which communicates with the VAT Service Orchestrator API.

3.3.1.2.2 Description of components

The components comprising Vulnerability Assessment Tools are described below.

Scan Configurator is a web interface for Vulnerability Assessment Tools. It enables users to configure and trigger vulnerability scans, set schedules for scanning tasks, review tasks' results, as well as create custom vulnerability detection scripts.

These scripts are stored in the **Custom Scanning Scripts Catalogue**. They can be written in any of the scripting languages, supported by the script interpreters included in the Registry. The Catalogue can also store script templates that need to have some missing parameters or code added before execution.

CC BY-SA

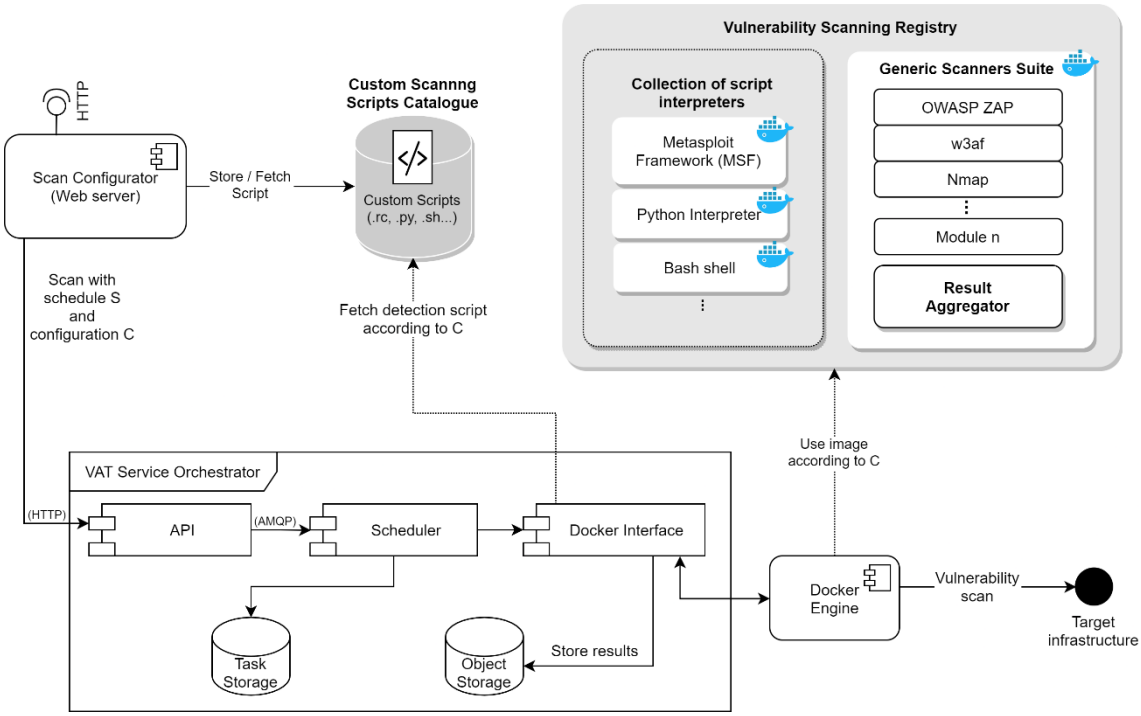

Figure 8. Internal architecture schema of Vulnerability Assessment Tools

Vulnerability Scanning Registry is a collection of Docker images for running vulnerability scans. It contains a **Generic Scanners Suite** image with several integrated scanning modules and a **Result Aggregator** component that combines results of the scanning modules into a single JSON result that can be shown in the Scan Configurator UI in a user-friendly way. The integrated scanning modules are OWASP ZAP [12], w3af [11], and Nmap [13]. ZAP and w3af are web application vulnerability scanners. When a scan is launched against a targeted website, they use crawlers to scan the website and identify potentially vulnerable pages and endpoints. For the detection of injection vulnerabilities, they use crafted payloads in automatic queries and observe the server's output, searching for patterns that would indicate potential vulnerabilities. Several server misconfiguration weaknesses can also be detected. Nmap is a network reconnaissance tool with vulnerability scanning capabilities. It can detect devices on the network and servers (listening ports) running on them, identify versions of the running servers and use various scripts to remotely detect specific vulnerabilities.

Beside the Generic Suite, it also holds several Docker images [14] of script interpreters that can run user-provided custom scanning scripts. This **Collection of script interpreters** can be used to detect specific vulnerabilities, to monitor uptime and availability of services, or for other repetitive tasks. Three interpreters are currently included: Metasploit Framework [15] (running Metasploit resource scripts and Metasploit modules written in Ruby), Python, and Bash. The Scanning Registry is designed in a modular way, so that additional scanners or script interpreters can be added easily.

The functionality of custom scanning scripts can be used to gather evidence about any metric obtainable by executing some code. For example, such scripts can access a CSP-internal API, check whether some server is available, search for errors in logs obtained from another server, etc. The script should return the result either in its exit code or as contents of a file. The result is then used by VAT to construct an evidence object with the pre-configured metric identifier.

VAT Service Orchestrator contains several subcomponents responsible for scheduling and orchestration of scans, as well as communication with other components. Internal

CC BY-SA

communication between components is realized through the AMQP protocol by a RabbitMQ [16] server (not shown in the figure for clarity). The Scan Configurator server communicates with the core components through the API, which also provides authentication and authorization capabilities. The API component is also accessed by the Wazuh & VAT Evidence Collector component (see Figure 7), which generates evidence objects according to the configuration and results of VAT and forwards them to the Security Assessment component (part of Clouditor) to be in turn processed by other MEDINA components.

Scheduler is the component responsible for triggering scanning tasks according to their configured schedules. Task Storage database is used to store the schedules and configurations. When a specific task is triggered, it communicates its configuration to the Docker Interface component that prepares the required files and parameters and executes the container spawned with the respective Docker image from the Registry in the Docker Engine. The Docker Interface also retrieves results of the finished scanning tasks and stores their output files in the Object Storage database, from where it can be retrieved by users.

3.3.1.2.3 Technical specifications

The various subcomponents of VAT use different programming languages, frameworks, and libraries. The backend components are mostly written in Node.js, except Scheduler which is written in Go. MongoDB [17] is used for the Task Storage, and OpenStack Swift [18] for the Object Storage and storage of custom scanning scripts. Scan Configurator frontend is built with the Angular [19] web framework.

The Generic Scanning Suite is built as a single Docker image with Ubuntu as base image with required scanning modules installed (OWASP ZAP [12], w3af [11], Nmap [13]). Cscan package (part of the open-source Faraday vulnerability scanning platform [20]) along with several additional Python and Bash scripts are included for triggering the scanning modules according to configuration parameters. The Result Aggregator is written in Python and outputs a JSON file containing outputs of all the scanning modules used.

The custom script interpreters are separate Docker images with the respective tools.

The communication among the core components is carried out with AMQP through a RabbitMQ [16] server. The API component exposes an HTTP REST API invoked by the Wazuh & VAT Evidence Collector (which is written in Python and was already described in Section 3.2).

3.3.2 Delivery and usage

3.3.2.1 Package information

The code of Vulnerability Assessment Tools is structured in several Git repositories according to the components described above. All components are packaged as Docker images.

The component acting as a bridge of Vulnerability Assessment Tools to MEDINA, Wazuh & VAT Evidence Collector, is described in Section 3.2.2.1.

3.3.2.2 Installation instructions

Deployment scripts are provided using Vagrant [21] and Ansible [22] in the “vat-deployment” repository. A single virtual machine is provisioned running all the necessary services for VAT.

To run the demo deployment process, simply clone the repository and run:

```
make create provision
```


CC BY-SA

3.3.2.3 User Manual

By navigating an internet browser to the IP address of the management machine, the user can access the VAT configuration portal, review the demonstrative vulnerability scans, or create new scanning tasks.

3.3.2.4 Licensing information

Vulnerability Assessment Tools framework is licensed as proprietary, Copyright by XLAB.

The Generic Scanning Suite, a containerized component integrating several vulnerability scanners, is developed by XLAB and open-sourced with the Apache Licence 2.0.

Several sub-components used as part of VAT are open source:

- OWASP ZAP (Apache Licence) [12]
- w3af (GPLv2) [11]
- Nmap (modified GPLv2)[8] [13]
- Faraday (GPLv3) [20]
- Metasploit (BSD) [15]

Wazuh & VAT Evidence Collector, developed by XLAB, is available open source (Apache Licence 2.0).

3.3.2.5 Download

VAT deployment demo repository is available at MEDINA's GitLab:

https://git.code.tecnalia.com/medina/public/vat-deploy

The Generic Scanning Suite source code repository is also available in MEDINA's GitLab:

https://git.code.tecnalia.com/medina/public/vat-genscan

Due to proprietary licensing, other parts of the VAT framework are hosted on XLAB's internal GitLab. The code can be made available upon request.

Source code of the individual included scanners can be found in their respective project repositories.

Source code of the Wazuh & VAT Evidence Collector is available in a separate repository:

https://git.code.tecnalia.com/medina/public/wazuh-vat-evidence-collector

3.3.3 Advancements within MEDINA

Vulnerability Assessment Tools were developed in a previous H2020 project, CYBERWISER.eu [23]. In that project, the VAT framework was used for the detection of vulnerabilities as well as for the scheduling of various actions connected to defence and attacks of infrastructure in a controlled and enclosed cyber range environment.

In the initial phase of MEDINA, an analysis was made to determine the EUCS requirements that are feasible to be verified or satisfied by VAT. Later, the internal architecture of VAT was restructured, and the deployment scripts were rewritten to support the deployment in a general

[8] https://nmap.org/npsl/

(cc) BY-SA

(cloud) environment instead of the specific cyber range setting. The APIs were adapted to be prepared for the integration with the MEDINA components. The Evidence Collector component was developed with a specific module to interact with VAT and produce relevant evidence.

In the scope of MEDINA, the following advancements have been made so far:

- Analysis of the EUCS requirements, feasible to be verified or satisfied by VAT.
- Restructuring of the internal VAT architecture.
- Deployment scripts adapted and rewritten to support deployment in a general (cloud) environment.
- Adaptation of APIs.
- Wazuh & VAT Evidence Collector developed with the connection to Clouditor (Security Assessment) and a VAT-interface module.
- Adaptation of the web interface and authorization modules.
- Implementation of metric identifiers in tasks to enable generic evidence collection (for user-provided metric ID).

3.3.4 Limitations and future work

As described above, VAT is composed of multiple modules: several vulnerability scanners and a framework for running custom, user-defined evidence collection scripts. Confidence of the evidence gathered with VAT can vary greatly depending on the VAT module used and the definition of a specific metric. The generic vulnerability scanners (e.g., w3af, OWASP ZAP) are primarily designed to be used in manually guided penetration tests. Thus, vulnerability detection results can often contain false positives that should be analysed by an expert. Evidence collected solely based on the results of such results therefore cannot be regarded with full confidence.

On the other hand, there are considerably less errors when a vulnerability detection tool is configured to check for the presence of a specific vulnerability (e.g., Nmap or Metasploit script). The accuracy of custom (user-provided) scripts entirely depends on their implementation, in this case VAT is only used as a framework for running such scripts and packaging and forwarding the results as evidence.

Some requirements of the EUCS standardisation framework require the CSP to have vulnerability or malware detection tools deployed on certain systems and to monitor their results. By using the vulnerability scanning capabilities of VAT (combined with monitoring of the results), the CSP effectively satisfies such requirements for their cloud service. In this case, the automatically obtained evidence refers to the functioning of VAT, which can be managed effectively and monitored with high confidence.

In the current state of implementation (in month 24), VAT can be used to produce evidence based on user-provided custom scripts. In theory, evidence can be obtained for any metric, but the confidence of such evidence depends on the user-provided code. The generic vulnerability scanners can be run with VAT, but there is currently no evidence being generated from the results of these scans. Future work will be focused on the implementation of metrics for the updated EUCS requirements in scope of MEDINA that concern vulnerability scanning compliance, preparation of examples for evidence gathering using user-provided scripts, and finalizing the integration with other components.

CC BY-SA

4 Security Assessment of Cloud Applications

This section presents the MEDINA components related to estimating the security of cloud applications and collecting evidence based on the analysis of their source code. The functionalities and implementation of the two components under development in Task 3.3 (Cloud Property Graph and Codyze) are described in the following subsections.

4.1 Cloud Property Graph

The Cloud Property Graph (CloudPG) is a further tool, developed within the first year of the MEDINA project and improved and extended in the second year. It combines static source code analysis with cloud infrastructure analysis. To that end, a library for static code analysis, the cpg[9], has been extended with analysis logic for cloud workloads. The implementation of this tool is being developed in an open-source repository on GitHub[10].

One problem the CloudPG addresses is that isolated security analysis on workload- or source code-level can result in many false positives: for example, authorization or encryption requirements may be implemented either on the infrastructure- or source code-level, thus both levels have to be analysed in combination to allow for a comprehensive assessment of, e.g., authorization or encryption requirements.

4.1.1 Implementation

4.1.1.1 Functional description

The cpg library, which forms the basis of the CloudPG, creates a property graph of source code that is enhanced by the CloudPG with information about the current resource configurations. Also, data flows between resources are added to the graph, e.g., HTTP requests between microservices for an excerpt of a graph generated by the Cloud Property Graph. Figure 9 shows several nodes and edges introduced by the CloudPG, for example security properties (based on the cloud resource ontology), like authenticity and transport encryption (see top left), and HTTP calls: the POST node describes a HTTP POST request to (TO edge) a HTTP endpoint that in turn has a certain path as its PROXY_TARGET.

This way, it is possible to identify security problems in the intersection between infrastructure and source code. For example, it can be detected if logging functionalities are implemented and if yes, if the logs are stored in an allowed region. This combined reasoning would be more difficult to do when assessing isolated evidence about source code and cloud workloads.

To also enable the detection of privacy threats, the CloudPG implementation has been extended with the following features: improved detection of data flows in HTTP connections, taint tracking via dedicated labels, detection of cryptographic libraries (e.g., for cryptographic signatures), detection of database operations, and more.

Since this combined analysis requires a common model of how, e.g., logging functionalities are implemented, and what they are called in different cloud systems, the CloudPG again makes use of the cloud resource ontology presented in deliverable D2.4 [5], and the security properties it defines. Consequently, security-relevant concepts can be analysed across source code and infrastructure configurations.

[9] https://github.com/Fraunhofer-AISEC/cpg
[10] https://github.com/clouditor/cloud-property-graph/

CC BY-SA

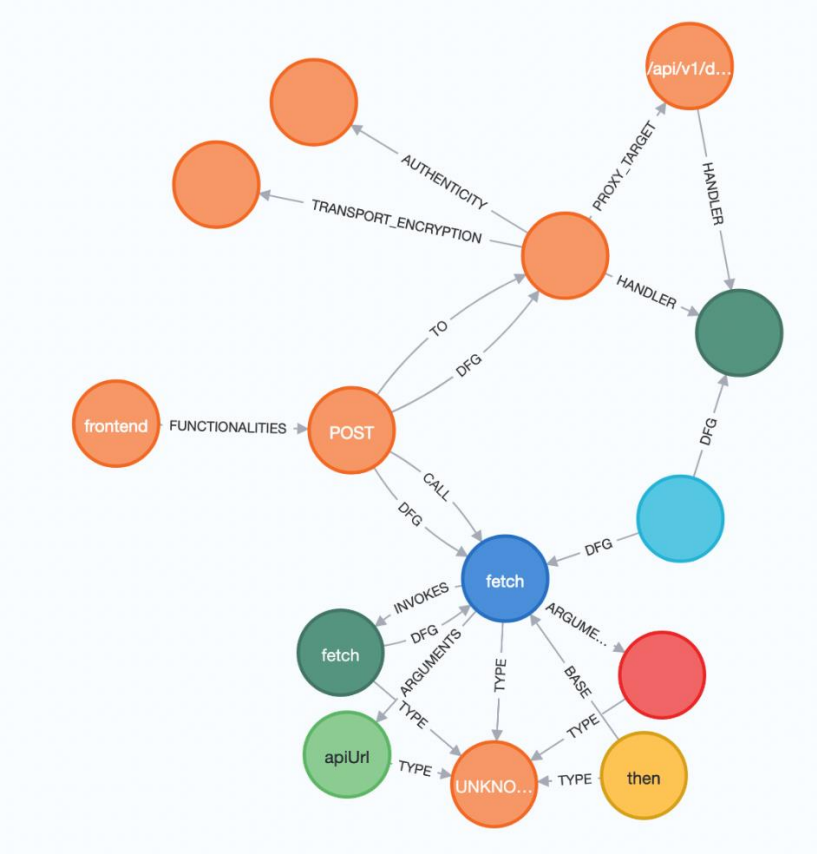

Figure 9. An excerpt from the graph generated by the Cloud Property Graph

Related requirements

The relevant requirements from the deliverable D5.2 [4] are listed below with a brief description of how they are implemented.

Requirement id	TEGT.S.02
Short title	Collect evidence from source code via CPG
Description	The developed tool must be able to parse the source code of cloud applications written in different programming languages and transform into the agnostic representation of the CPG.
Implementation state	Fully implemented

Requirement id	TEGT.S.03
Short title	Implement information and data flow analysis
Description	The developed tool must be able to perform information and data flow analysis on a cloud application.
Implementation state	Fully implemented

Similar to Codyze, the CloudPG is able to analyse source code regarding data flows, control dependence, and other properties.

CC BY-SA

Requirement id	TEGT.S.10
Short title	Connect infrastructure- and application-level security analyses
Description	The developed tool should be able to bridge the gap between infrastructure- and application-level security analysis by extending graph-based code analysis to the cloud resources, allowing to identify data flows across cloud resources.
Implementation state	Fully implemented

This requirement describes the core idea of the CloudPG: by combining both source code analysis and deployment information, an accurate assessment of security properties can be made.

Requirement id	TEGT.S.07
Short title	Support for common programming languages, libraries, cloud services
Description	The developed tool should support common programming languages, libraries and cloud services. Support for all programming languages, libraries and cloud services is infeasible.
Implementation state	Fully implemented

Similar to Codyze, the CloudPG supports multiple programming languages, including Java, C, Python, Go, and Typescript.

Note that the common requirements (TEGT.C.X) are currently not relevant for this tool since it is a novel research concept, whose integration into the framework still needs to be designed.

4.1.1.1.1 Fitting into overall MEDINA Architecture

The CloudPG is a separate component implementing evidence gathering and assessment. As it is a new research approach, it is not integrated with other components. Two possible approaches exist for its integration: First, its output may be adjusted to provide evidence in the required MEDINA format to the Clouditor Security Assessment. In this case the CloudPG's graph-based analysis results would have to be transformed to the respective evidence format. Second, it can be extended with a custom assessment service which is then integrated with the Orchestrator. This latter approach has the advantage of leaving more room for custom assessment logic but may require more effort.

4.1.1.2 Technical description

4.1.1.2.1 Prototype Architecture

The CloudPG's architecture is based on the architecture of the underlying cpg library. First, it uses the cpg library to build a code property graph of the given source code. It then applies custom *passes*, i.e., extendible modular analysis logic, to analyse properties of the code and its deployment that are relevant in the context of (cloud) security. This added information is then introduced in the graph to make it accessible for manual analysis and possibly automatic analysis applications.

Some examples of such custom passes are presented in the following:

- HTTP calls: The CloudPG analyses code to detect HTTP calls between microservices and adds edges to the graph between the respective nodes, e.g., from a code entity that uses an HTTP framework to realize the HTTP call to the respective HTTP endpoint. HTTP

CC BY-SA

endpoints are identified in another pass which is able to detect these in the Spring framework for Java, the Flask framework for Python, as well as the Gin framework for Go.

- Logging: The CloudPG detects logging functionality, such as the zerolog[11] library for Go.
- Deployment information: The CloudPG detects GitHub workflow files in a project, which specifies where the code is deployed, e.g., as Docker containers in a Kubernetes cluster, and adds configuration information about the deployment environment.
- Databases: The CloudPG identifies connections and other operations related to databases, such as MongoDB and PostgreSQL.
- Label extraction: In the context of improvements of the CloudPG for privacy analysis, labels have been added to mark personal data (or otherwise sensitive data), which can be extracted by a dedicated pass to track their flow across the application.

4.1.1.2.2 Description of components

The CloudPG is not divided into separate components. A separate assessment component may be developed in the future. It does, however, employ an easily extendible structure for additional passes.

4.1.1.2.3 Technical specifications

The CloudPG is written in Kotlin. As described above, it makes use of the cpg library to build a basic code property graph.

Please note that the user manual and installation instructions of CloudPG can be found in Appendix D.

4.1.2 Delivery and usage

4.1.2.1 Package

The tool is not yet available as a Docker image. It currently needs to be installed as described in Appendix D.

4.1.2.2 Licensing

The tool is licensed under the open-source Apache License 2.0.

4.1.2.3 Download

The project is available open source on GitHub: https://github.com/clouditor/cloud-property-graph/

4.1.3 Advancements within MEDINA

The Cloud Property Graph is based on the cpg, which is a project developed independently of MEDINA. The CloudPG's additions described above, however, have completely been developed within the MEDINA project. In its approach of combining source code analysis with infrastructure analysis, it complements Codyze (see Section 4.2).

In the second iteration of the MEDINA components' development, CloudPG has been extended with dedicated privacy analysis functionalities. To that end, the various privacy goals defined in the LINDDUN framework [24] have been analysed, operationalized, and translated to code

[11] https://pkg.go.dev/github.com/rs/zerolog

CC BY-SA

properties. Finally, respective passes have been integrated, e.g., for database connections, specific HTTP connections, privacy labels, and more. Also, a testing library has been developed for the evaluation of the component.

A scientific paper about the tool and the described approach has been published at the IEEE International Conference on Cloud Computing 2021 (CLOUD) [25]. A further publication about extensions of the CloudPG for privacy analysis is in review at the time of writing.

4.1.4 Limitations and future work

The approach implemented in the Cloud Property Graph has some general limitations. First, it is constrained by the available source code and accessible APIs, i.e., when libraries are used whose source code is not available, or source code is not available for other reasons, it will not be part of the resulting graph and cannot be analysed for security problems. Regarding the APIs, the limitation is the same as for the evidence collection with Clouditor: only the information the cloud APIs offer can be analysed. Second, the approach currently generates additional manual effort since the tool has to be set up, it has to be manually applied, and its results need to be manually evaluated. However, its application and result analysis have potential for automation which should be addressed in future work. Also, its integration with the MEDINA framework, i.e., with the Security Assessment or Orchestrator should be addressed in future work.

4.2 Codyze

Codyze is an open-source static application security testing tool. Its main goal is to verify if application source code complies to security requirements. Security requirements are derived from security requirement catalogues such as ENISA EUCS [1]. Security requirements are broken down into checkable source code properties. Afterwards, Codyze verifies specified source code properties and thereby can provide evidence and assessment results if a requirement is sufficiently realized in software.

Codyze supports security by design. It can recognize potential security flaws violating compliance to security standards like ENISA EUCS. It provides early feedback during the development process and can ensure that less security flaws remain in a production-ready deployed cloud service. It can also act as a quality gate within an CI/CD pipeline and prevent that cloud services are deployed in production which don't meet defined compliance requirements. It supplements the MEDINA framework by ensuring that consumable cloud applications and services are implemented securely.

4.2.1 Implementation

4.2.1.1 Functional description

Codyze uses the MARK DSL [26] to specify checkable software properties. MARK can model entities and define rules that must hold for the usage of an entity. Codyze evaluates MARK rules against provided source code and attest if a rule is adhered to or not. Based on the evaluation result from MARK rules, software properties required to fulfil security requirements are validated.

Currently, Codyze can analyse source code written in C/C++ and Java. Moreover, it ships with MARK rules for cryptographic libraries Bouncy Castle for Java and Botan for C++. Thus, source code can be checked if cryptographic operations with Bouncy Castle or Botan are properly implemented and thereby attest state-of-the-art cryptography of sufficient strength.

CC BY-SA

As Codyze analyses source code, it is not integrated into the cloud platform itself. It is a tool used by CSPs to validate the source code of applications and services prior to deployment and general availability in the cloud. Therefore, Codyze must be integrated into the CSP's development, continuous integration, and continuous deployment pipeline. Once integrated, Codyze can check submitted code while it is being developed. Configured as a breaking check point in a CI/CD pipeline, it can prevent the roll out of software not meeting security requirements.

Codyze submits results from its analysis to the Orchestrator. It uses the MEDINA data model to send evidence. Afterwards, Codyze posts its assessment results referencing the previously submitted evidence.

In addition, Codyze creates a report in the SARIF format [27]. SARIF is an OASIS standard that specifies a format to encode information and findings from static code analysis into an exchangeable format. This report is saved during a CI/CD pipeline and can be reviewed afterwards to identify problems and fix them. Thereby, code repository platforms with integrated CI/CD functionality such as GitHub[12] can automatically process SARIF reports and represent the respective information integrated on their platform. Developers can use this information to fix problems in their source code.

Related requirements

The relevant requirements from D5.2 [4] are listed below and a brief description of how they are implemented is given.

Requirement id	TEGT.C.01
Short title	Continuous collection
Description	The developed tools must be able to collect evidence continuously, i.e., in (high)-frequency intervals.
Implementation state	Partially implemented

Codyze is integrated into the CI/CD pipeline at CSPs. It is executed based on the frequency of committed code changes.

Requirement id	TEGT.C.02
Short title	Provision to defined interfaces
Description	The developed tools must provide collected evidence to a security assessment tool via its offered APIs.
Implementation state	Fully implemented

Codyze submits all evidence to the Orchestrator. As Codyze provides its own assessment results, evidence is submitted for reference in the resulting assessment result.

Requirement id	TEGT.S.03
Short title	Implement information and data flow analysis
Description	The developed tool must be able to perform information and data flow analysis on a cloud application.

[12] https://docs.github.com/en/code-security/code-scanning/integrating-with-code-scanning

CC BY-SA

Implementation state	Fully implemented

Codyze can perform information and source code analysis; the extended analysis for contextual information of cloud workloads has been addressed in the Cloud Property Graph tool which is closely related to Codyze. For example, it can analyse infrastructure configurations and CI/CD information of respective configuration files to check where a certain piece of code is deployed in a cloud service.

Requirement id	TEGT.S.04
Short title	Support expression of security requirements
Description	The developed tool must be able to support the expression of security requirements to be checked on application code. Requirements come for example from WP2.
Implementation state	Partially implemented

While Codyze can verify security requirements, defined in the MARK DSL, it is not yet able to verify MEDINA-related requirements, e.g., written in Rego. Instead, requirements are mapped to MARK rules such that validation of MARK rules indicates compliance to requirements.

Requirement id	TEGT.S.05
Short title	Verify security requirements
Description	The developed tool must be able to verify security requirements and raise warnings/errors with respect to secure coding practices and secure information and data flows.
Implementation state	Partially implemented

Codyze is currently able to generate warnings for identified non-compliances. It remains to integrate these warnings in MEDINA, e.g., in a user interface. The current rule set needs to be extended.

Requirement id	TEGT.S.06
Short title	Retrieve source code of cloud applications
Description	The developed tool should be able to retrieve (semi-)automatically the source code of cloud applications requiring analysis.
Implementation state	Partially implemented

Source code is provided as part of the CI/CD pipeline. The fulfilment of this requirement will be validated during field tries with partners.

Requirement id	TEGT.S.07
Short title	Support for common programming languages, libraries, cloud services
Description	The developed tool should support common programming languages, libraries and cloud services.
Implementation state	Partially implemented

CC BY-SA

Support for all programming languages, libraries and cloud services is infeasible.

Requirement id	TEGT.S.08
Short title	Provision of malware, intrusion, and vulnerability detection tools
Description	Tools for malware detection, intrusion detection, and vulnerability scanning must be provided to assist CSPs with satisfying related requirements of security standards or to verify the compliance with such requirements.
Implementation state	Partially implemented

Codyze is provided as a binary distribution and as a container image. It can be integrated as a validation step in CI/CD pipelines. The integration, usage and suggested configuration will be documented based on the environment used by partners.

4.2.1.1.1 Fitting into overall MEDINA Architecture

Codyze integrates itself into the overall MEDINA architecture as an application-level evidence collection and security assessment tool (see Figure 1 in Section 2). Codyze assesses source code of cloud application and ensures compliance to security requirements catalogues like ENISA EUCS within applications. It submits assessment results to the Orchestrator for further processing. In addition, it stores the evidence used to derive an assessment result with the Orchestrator to verify provenance.

4.2.1.2 Technical description

4.2.1.2.1 Prototype architecture

Codyze consists of an executable binary distribution and runs stand-alone. It is also available as a container image. Codyze is executed on the source code of cloud application and services. Therefore, there are no server components or agents. Figure 10 depicts Codyze architecture.

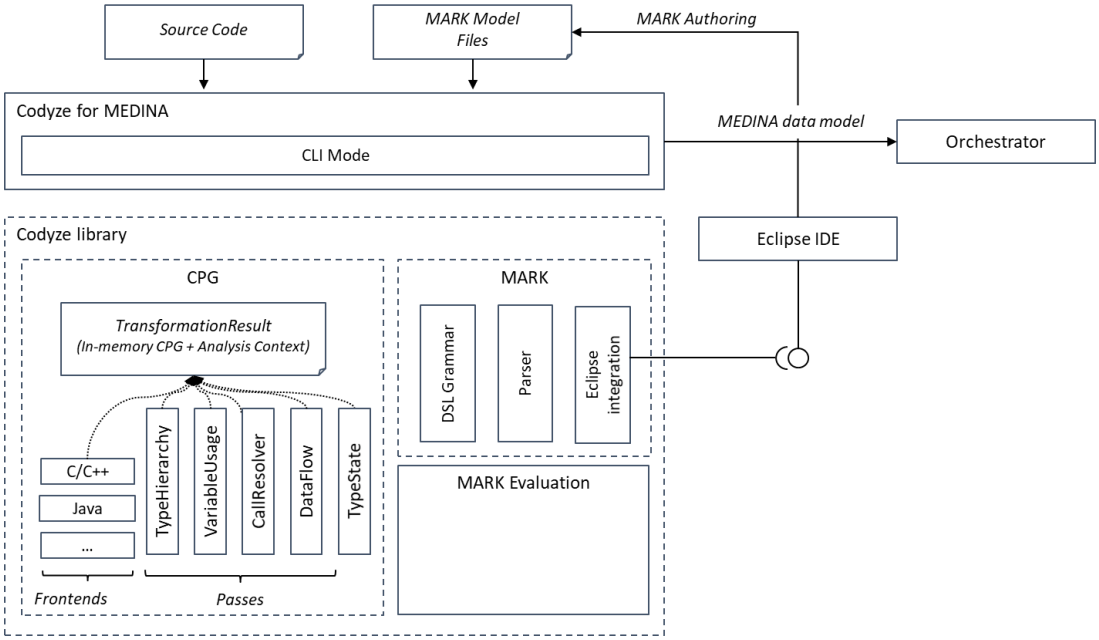

Figure 10. Codyze architecture

Codyze for MEDINA provides a command line interface. This is the main interface to run Codyze automatically in a CI/CD pipeline. In this mode, Codyze generates a report that contains

CC BY-SA

problematic source code locations where security requirements are not met. In addition, this mode will return an error code when security requirements are not met and can thus terminate CI/CD pipelines. This behaviour ensures that Codyze prevents the roll out of cloud applications and services that do not comply to security requirements as required by catalogues like ENISA EUCS.

Internally, Codyze uses the cpg library. This library implements a code property graph. This graph is a multigraph representing source code structures in a graph representation. On this graph model of the source code, Codyze can perform the source code evaluation.

The evaluation is specified in MARK. MARK files are provided to Codyze either as part of Codyze or as a path to MARK files on the command line. These MARK files are parsed by Codyze and define the necessary evaluation steps to validate the compliance to security requirements.

The results of the evaluation are provided to developers as SARIF reports. In addition, findings are submitted to the Orchestrator of the MEDINA framework in the specified data format.

4.2.1.2.2 Description of components

Codyze for MEDINA makes use of the following components:

- **MARK** is a domain specific language to specify verifiable properties that source code must adhere to. It can, for example, restrict possible data values and their flow, or specify interactions between objects. A corresponding software library built on top of Xtext[13] provides the language grammar and parser functionality. In addition, a generated Eclipse plugin provides MARK specific editing support in Eclipse IDE.
- **cpg** is a library implementing a code representation based on the concept of a code property graph [28]. It is responsible for parsing source code and providing a graph-based code representation suitable for querying code properties.
- **Codyze library** provides the analysis engine for Codyze. It uses the CPG to parse source code. In addition, it uses the MARK library to parse MARK files. The Codyze library implements the analysis steps to interpret MARK rules and identify rule violations in source code. Assessed rules generate a finding that either certifies compliance or documents a rule violation.

4.2.1.2.3 Technical specifications

Codyze is developed in Java and Kotlin. It uses the libraries CPG and MARK as dependencies. In addition, Codyze ships with MARK rules that check compliance to strong, state-of-the-art cryptography.

The integration of Codyze at the CSPs requires a platform for CI/CD. Codyze can be integrated into CI/CD pipelines either by using the binary distribution or the container image as a step during validation step of the pipeline.

Codyze should be configured as a static application security test. It should prevent successful CI/CD pipeline completion if violations are discovered. It thus prevents the deployment of artefacts into production that do not meet minimum compliance requirements.

In addition, Codyze generates a report in SARIF format. SARIF is a standardized description of findings from static analysis tools. Its adoption is rising and many SAST tools now include support

[13] https://www.eclipse.org/Xtext/

CC BY-SA

for SARIF. This is also reflected in platforms for source code management and CI/CD integration. Platforms like GitHub support the evaluation of SARIF reports and include results within the UI of their platform. Developers can identify problematic results on these platforms for example as code annotation in merge requests.

Please note that the user manual and installation instructions of Codyze can be found in Appendix C.

4.2.2 Delivery and usage

4.2.2.1 Package information

Codyze is packaged as a binary distribution in a ZIP archive. In addition, the public code repository contains a Dockerfile to build a container from the source code.

4.2.2.2 Licensing information

Codyze for MEDINA and its components are licensed under the open-source Apache License 2.0.

4.2.2.3 Download

Codyze for MEDINA is available from the public MEDINA GitLab repository:

- https://git.code.tecnalia.com/medina/public/codyze

The Codyze source code for the Codyze library is available from its GitHub repository:

- https://github.com/Fraunhofer-AISEC/codyze

The MARK source code is available from its GitHub repository:

- https://github.com/Fraunhofer-AISEC/codyze-mark-eclipse-plugin

4.2.3 Advancements within MEDINA

Codyze has been successfully adopted for MEDINA. The original Codyze library has been extended to support the analysis of source code commonly seen in cloud service implementations. The analysis engine within Codyze has been improved.

Secondly Codyze has been integrated into the MEDINA architecture. It supports the MEDINA data model. Findings from Codyze are submitted as evidence and assessment results with the Orchestrator. Thus, results from the analysis of Codyze can be reviewed within the MEDINA framework.

Thirdly, new MARK specifications have been written that model common cloud service functionality. These models have been used to define MARK rules that support compliance to EUCS requirements. The mapping between rules and requirements has been initially implemented. Currently, the recently developed MARK specifications cover transport encryption using TLS in support of EUCS control CKM-02 Encryption of data in transit.

4.2.4 Limitations and future work

Codyze analyses source code and its usefulness are therefore limited by the inputs it gets: Since there is no reliable source for knowing which code exists and should be deployed, it is also not possible to verify within Codyze if all relevant code has been analysed. Therefore, we assume that Codyze is applied to all relevant code.

CC BY-SA

Another limitation of Codyze is the use of MARK as a specification language. Source code properties need to be modelled and rules need to be specified. The resulting MARK specification is to some extend specific a programming language and modelled software library. Hence, Codyze can analyse and assess only source code for which specifications exist. Moreover, specifications need to be updated to keep up to date with changing requirements and updated software libraries.

The main work within the following project period is to extend the modelled use cases in MARK and the coverage of the EUCS. This work will be done in close cooperation with project partners that develop cloud service. The minimal goal is a catalogue of MARK specification that supports our partners in validating compliance to EUCS in the source code of their product.

(cc) BY-SA

5 Assessment of Organisational Measures

Assessment of organisational measures is handled by the MEDINA component named AMOE (Assessment and Management of Organisational Evidence), which is designed to extract evidence from policy documents. Furthermore, it allows to set and submit assessment results to the MEDINA framework.

5.1.1 Implementation

5.1.1.1 Functional description

AMOE is a proof-of-concept prototype for assessment and management of organisational evidence. It is designed to extract evidence based on organisational metrics, targeted to specific parts of policy documents. After extraction, the evidence can be inspected in the GUI of the tool. Users can then decide on the compliance status of a metric. Once the user is satisfied with the assessment, the result and evidence can be forwarded to the Orchestrator. While extracting the evidence is fully automated, the final decision is made by the user.

To improve AMOE, different evidence extraction methods are researched, the main one is depicted in Figure 11. After uploading via GUI or API, a policy document it goes through the various stages of the extraction pipeline. There are two stages, the first is the pre-processing, the second the actual evidence extraction.

Pre-processing

The PDF document is transformed into a HTML with poppler utils'[14] pdftohtml. While it is processed, common errors for section headings are fixed. The result is a structured document that eases the filtering process. In the last stage of the pre-processing, information such as table of contents, or parts of the header or footer are removed. The whole process depends heavily on the quality of the PDF, which is reflected in the results.

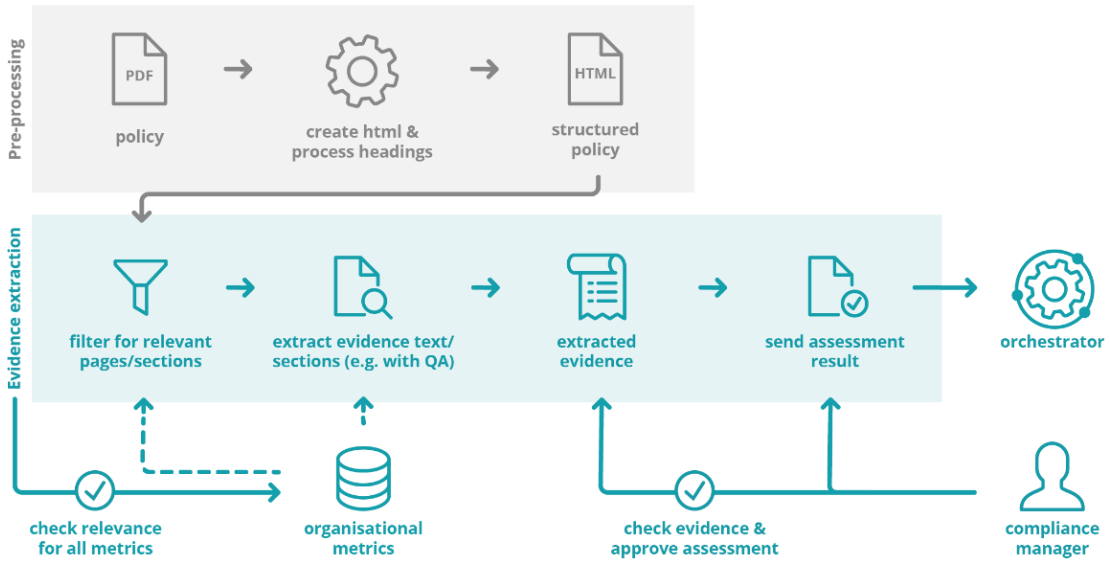

Figure 11. Main architecture for AMOE (keyword-based extraction method)

[14] Library for pdf transformation/rendering: https://poppler.freedesktop.org

CC BY-SA

Evidence extraction

After preparation of the document, a subprocess is started for every organisational metric. First, the search space is reduced by filtering the document for relevant sections/paragraphs. This is done using the keywords defined in the metric and the section headings marked in the pre-processing. Second, the metric question as well as the filtered text is fed into the pre-trained question answering model. Third, extracted answer/evidence is marked in the HTML document and the evidence is stored.

In the case a target value is defined for an organisational metric, AMOE derives an assessment hint. This hint is using the extracted evidence and compares it with the defined target value. As some metrics are phrased as an open question, not all have a target value defined. The main goal of AMOE is to aid the user, not perform blind assessment. Therefore, the results need to be verified by a human.

At this stage, the data is available for a user (i.e., compliance manager) to inspect the result via the GUI or API and set the final assessment result. Once set, the assessment result can be sent to the Orchestrator.

Related requirements

The relevant requirements from Deliverable D5.2 [4] are listed below with a brief description of how they are implemented.

Requirement id	OEGM.01
Short title	Continuous collection of organizational evidence
Description	The developed tool using NLP must be able to collect organizational evidence.
Status	Fully implemented

Evidence is automatically extracted after a file has been uploaded.

Requirement id	OEGM.02
Short title	Provision to defined interfaces
Description	The developed tool using NLP must provide collected evidence to the central evidence collection component (T3.1) via its offered APIs.
Status	Fully implemented

The users of the component can forward an assessment result (and evidence as needed by the API) to the Orchestrator. This can be triggered by the UI or API once a compliance status has been set for a metric or an extracted evidence.

Requirement id	OEGM.03
Short title	Usability for auditors
Description	The evidence management component should provide easy-to-use functionalities for auditors to search through relevant evidence. The assessment is handled manually though the UI. The assessment can be adjusted via API (should be checked/verified by a human beforehand).
Status	Fully implemented

AMOE provides the extracted results in an interactive UI. Evidence is highlighted in the extracted answer as well as in the processed document. Furthermore, it extracts the page number where the evidence was found so one can double check the information in the original uploaded

CC BY-SA

document. The computed assessment hints are designed to help users in reaching their decisions. As they might be false, the final decision on an assessment is done by the user.

Requirement id	OEGM.04
Short title	Minimum evidence storage
Description	The evidence management component must be able to store and provide evidence at least back to the last assessment (if needed).
Status	Fully implemented

The uploaded and extracted data is stored until it is manually deleted. It can be deleted via the GUI. Log information can be deleted by an administrator with access to the database.

Requirement id	OEGM.05
Short title	Evidence Assessment results
Description	The assessment results of evidence assessments must be submitted to the evidence Orchestrator via the API it provides.
Status	Fully implemented

The assessment results can be forwarded to the Orchestrator using the API or GUI. See also Req. OEGM.02

5.1.1.1.1 Fitting into overall MEDINA Architecture

AMOE provides the functionality to add assessment results of organisational requirements/metrics to the MEDINA framework. It works with organisational metrics from the Catalogue of Controls and Metrics and accesses the predefined target values from the Orchestrator API (metric configuration). Alternatively, the metrics can be read from a local file. Once an uploaded file is processed and the evidence is processed and confirmed by a user, it can be forwarded to the Orchestrator and further according to the evidence pipeline defined.

5.1.1.2 Technical description

5.1.1.2.1 Prototype Architecture

Figure 12 depicts the AMOE architecture. The prototype core is the webservice based on the Quart Python library[15]. The API and GUI are served by this central component. For the interaction with the rest of the MEDINA framework, there is a dedicated subcomponent incorporating the auto-generated Python clients to the APIs based on their OpenAPI specifications.

The webservice's session management uses a separately deployed Redis instance. The evidence and log information are stored in the separately deployed MongoDB instance using the dedicated functions.

Not directly part of the prototype, but core part of the evidence extraction research is the quality check functions and separately deployed Inception[16] instance. Inception can be used to annotate the data needed for the quality measurements.

[15] https://pypi.org/project/quart/
[16] Annotation tool for NLP: https://inception-project.github.io/

(cc) BY-SA

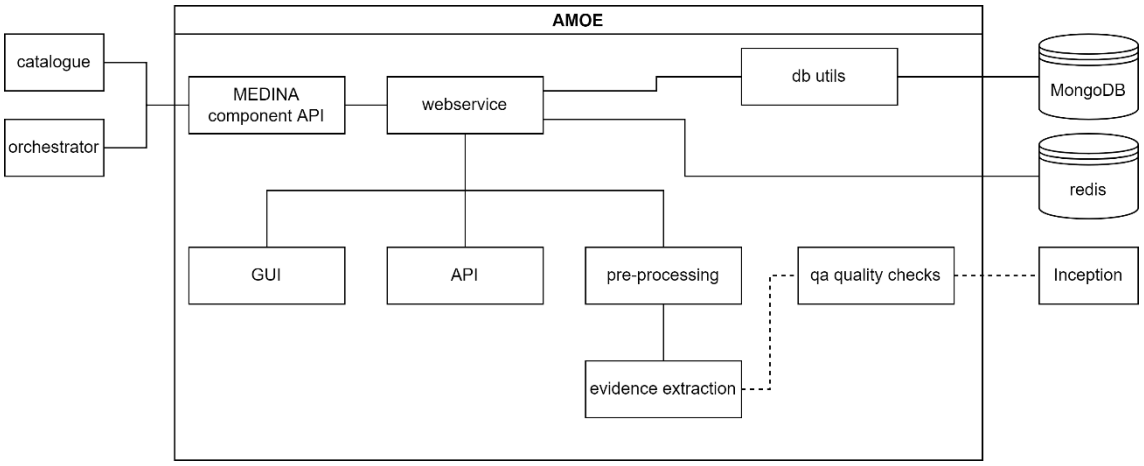

Figure 12. AMOE prototype architecture

5.1.1.2.2 Description of components

AMOE consists of a main webservice serving the GUI and the API. There are some parts that could be used independently of the main webservice. Figure 12 shows the main components of AMOE:

Webservice

This is the core component redirecting the data flow to the relevant subcomponents. It serves the GUI and API, and verifies authentication via the Keycloak[17] instance from MEDINA.

MEDINA component API

This subcomponent is used to access the different components from the MEDINA framework. This is used to retrieve requirements and metrics from the Catalogue of Controls and Metrics and the metric configuration from the Orchestrator. Furthermore, it is used for submitting the assessment results and extracted evidence.

DB utils

This subcomponent is used to store and access evidence results as well as local assessment results. It is also used to log relevant information such as by whom and when a document has been uploaded or an assessment result has been changed.

GUI

The graphical user interface serves to upload documents as well as to access the processed evidence. It enables the user to search, filter and manage the organisational evidence.

API

The API enables data access for other applications such as the company compliance dashboard. It can be used to perform the most essential functions of the GUI. These include uploading a document, retrieving the processed evidence, setting assessment results and submitting the assessment results to the Orchestrator.

[17] https://www.keycloak.org/

CC BY-SA

Pre-processing

This subcomponent is triggered in a background process once a document has been uploaded. It performs the necessary transformations to enable the evidence extraction.

Evidence extraction

This subcomponent is triggered after the pre-processing pipeline is done. It works as described in section 5.1.1.1.

5.1.1.2.3 Technical specifications

The tool is written in Python 3.x. It uses various Python libraries as well as the pdftohtml functionality from poppler utils[18]. The webservice is built on Quart[19], the evidence extraction is based on transformers[20], PyTorch[21] and the roberta-base-squad2[22] model from huggingface.

The component is using MongoDB[23] and Redis[24] to store the data. Evidence and logs are stored in the MongoDB. Redis is used in par with the quart-session library.

5.1.2 Delivery and usage

5.1.2.1 Package

AMOE can be deployed as a Docker container. Table 8 shows an overview of the repository folders and files.

Table 8. Overview of AMOE's source code package contents

Folder	Description
clouditor_evidence_client_legacy/	This folder contains the generated Python client for the evidence API of Clouditor based on their openapi file.
clouditor_orchestrator_client_legacy/	This folder contains the generated Python client for the Orchestrator API of Clouditor based on their openapi file.
inception_kubernetes/	This folder contains the kubernetes files to deploy a instance of the annotation tool inception
jenkins/	This folder contains the Jenkins pipeline code
kubernetes/	This folder contains the kubernetes files for deployment of AMOE.
metric_data/	This folder contains the local version of the metrics.
paragraph_extraction/	This folder contains the code for the pre-processing pipeline.

[18] https://poppler.freedesktop.org/
[19] https://pypi.org/project/quart/
[20] https://github.com/huggingface/transformers
[21] https://pytorch.org/
[22] https://huggingface.co/deepset/roberta-base-squad2
[23] https://www.mongodb.com/
[24] https://redis.io/

(cc) BY-SA

qa/	This folder contains the code for evidence extraction using the question answering model as well as code to compute quality scores.
static/	This folder contains the stylesheets and images for the webservice.
templates/	This folder contains the HTML templates for the webservice.
utils/	This folder contains code for utility functions of the webservice such as use of other MEDINA component's API, evidence extraction and database management.
/	The root folder contains the main webservice and configuration as well as the Dockerfile.

5.1.2.2 Installation

Clone the AMOE repository. Set up a MongoDB and a Redis instance (see kubernetes files in the repository).

Set the following fields in the config.py:

```
MONGODB_URL

KEYCLOAK_URL + authentification settings

CATALOGUE_API_URL

ORCHESTRATOR_API_URL
```

Run `hypercorn app:app -b 0.0.0.0:8000` to deploy the service locally, or deploy with kubernetes.

5.1.2.3 User Manual

AMOE user manual can be found in Appendix E.

5.1.2.4 Licensing

The component is planned to be licenced under GPLv2 or GPLv3.

5.1.2.5 Download

There are some licence issues to be addressed, so the AMOE component cannot be made public at the moment of writing this deliverable.

5.1.3 Advancements within MEDINA

The AMOE component has been developed from scratch for MEDINA by Fabasoft to cover evidence extraction for organisational requirements in the light of Task 3.4[25]. The advancements can be grouped into two parts – data management and the application itself.

Organisational metrics and annotation

[25] Development started late in the project as Fabasoft adopted this task from another partner. AMOE is foreseen as a Proof of Concept (PoC) in MEDINA.

(cc) BY-SA

Similar to the technical metrics, organisational metrics have been developed in cooperation with the domain experts of Bosch. For a subset of the organisational metrics, the relevant evidence has been annotated using the tool Inception.

Pre-processing pipeline

To enable evidence extraction on PDF documents, we developed a pre-processing stage, where the document is transformed into a HTML file. The information is reduced to what is deemed useful (e.g., removal of header and footer of the page; skip table of contents). Furthermore, only text that is deemed relevant is selected.

Evidence extraction

AMOE enables evidence extraction based on the organisational metrics. This works with standard NLP techniques and the use of a pre-trained question answering model.

UI development

To provide the information and functionality to a user, the webservice has been developed.

API development

To provide the information and functionality to a different client (e.g., the Company Compliance Dashboard [CCD], see D6.3, Appendix D – Workflows of Use Case 2), the API has been developed.

Quality checks

To get an impression of how well the prototype is working and to be able to develop / research for better extraction methods, quality checks have been implemented.

Currently, there are four basic approaches implemented for evidence extraction. The first one (keyword based) is described in the functional description in section 5.1.1.1 (Evidence extraction) above. The other approaches are quite similar; however, the order of the results differ. As they are still under development, the description is kept to a higher level in this paragraph. The main difference to the keyword-based approach is, that the same metric question is evaluated on every paragraph instead of once per filtered document text. The results are ordered in various ways, e.g., based on a similarity score (cosine similarity) between each paragraph and the keywords. The top result is selected to be extracted as evidence. The keyword-based approach performs best, therefore it is used in the deployment of the prototype.

Test cases have been constructed to determine the quality of the approach. Each test case is using a policy document and a set of organisational metrics for which the evidence has been annotated on the document. The annotations were edited using the Inception tool (see also section 5.1.1.2.1). The test case policy document is used for empirical and numerical analysis of the evidence extraction. In the rest of the section, details and results of the quality checks (score) and the two test cases are described.

Fabasoft test case:

Fabasoft created a specific document containing dummy policies for test purposes, as the internal could not be shared with the consortium. These policies are contained in a single document further referenced as “Fabasoft dummy policy document”.

Bosch test case:

(cc) BY-SA

Bosch has shared two policy documents, one of which contains more details and is therefore used for the experiments. The document used is further referenced as "Bosch IoT policy document".

Results:

Current results for the Fabasoft dummy policy document (Fabasoft test case):

- Keyword based approach: 19 / 28 = 0.68
- Score based approach: 13 / 28 = 0.46
- Similarity based approach: 7 / 28 = 0.25
- Similarity + score based approach: 13 / 28 = 0.46

Although, the main focus in research was on the metrics and policies by Fabasoft, here are the current results for the Bosch IoT policy document (Bosch test case):

- Keyword based approach: 10 / 50 = 0.20
- Score based approach: 6 / 50 = 0.12
- Similarity based approach: 8 / 50 = 0.16
- Similarity + score based approach: 9 / 50 = 0.18

28 organisational metrics have been annotated for the Fabasoft test case and 50 for the Bosch test case. The score is the ratio of the number of correctly retrieved text samples to the total number of the respectively annotated text passages ($score = \frac{\#\text{correctly retrieved evidence}}{\#\text{total annotated evidence}}$).

Some of the results can be explained by the difference of expertise between the annotators and the metric developer. However, we think the results can be improved by improving on the extraction pipeline, and performing curation on the annotated data and metrics. The metric questions and especially the linked keywords need to be revised.

5.1.4 Limitations and future work

The following sections describe current and future limitations of the AMOE proof-of-concept prototype.

5.1.4.1 Current limitations that will be overcome in next iterations during the project:

In task 3.4, the AMOE prototype is subject to limitations of the dataset (quantity and quality). Given that no suitable publicly available datasets were found, data needs to be constructed and adapted to the needs of the project. This is addressed by the creation of specifically designed organizational metrics designed on the basis of policies provided by Bosch and Fabasoft. For good results this requires time and expertise to map the relevant information in the policy text to concrete metrics that are specialized for extraction of this organisational information. This provided metrics and documents are the main input for the prototype and is used for building the evidence extraction pipeline as well as its evaluation measures. The organisational metrics can be revised and extended to increase the available dataset. Thus, we are able to get a broader picture on how well the prototype works in practice as well as whether some improvements to the evidence extraction pipeline are fruitful the way they are intended.

The quality of the current extraction results varies from sample to sample. However, with further research and improvement of the evidence extraction methods, this can be overcome up to a certain point. Confidence in the results of the extraction pipeline is measured by calculating how

(cc) BY-SA

much of the annotated data (ground truth) can be correctly extracted. This score can be used to validate added improvements to the extraction pipeline.

5.1.4.2 Generic limitations that will not be overcome (due to technology, specific requirements, etc.):

As the data for this task is rather limited, it was decided to use a pre-trained model. This model has been trained for question answering in a different domain, so it is not specific to the terminology of cloud service provider's policies. Empiric analysis has shown that the model provides reasonable answers for the given task, however, we suspect that the results could be improved with the roberta-base-squad2 model, given enough data. As we do not want to overfit on the little data acquired, we refrain from performing this step, to retain the generic output.

As the development and test environment by the MEDINA project do only include limited resources (no GPU, limited RAM and CPU), the processing of the policy documents takes quite some time. This can be partly overcome by using e.g., GPU for the question answering (it has been verified in a local test environment). However, with the use of GPUs this is not an instant process.

The AMOE component deals at the moment only with policy data in PDF. This is because it is hard to obtain enough generic evidence deemed organisational, without developing a solution that is too specific for a single cloud service provider. To gain a tool with high quality results for multiple providers, we focus on this task.

The evidence extraction process is limited to the organisational metrics. The used questions should be defined to retrieve a concrete answer / value to measure. No binary answers are possible with the chosen model and the model only selects a single answer. This could be extended in future work, for the moment it was deemed to be not in the scope as complexity increases and focus is on concrete evidence extraction.

There will be no fully automatic assessment; due to no guarantee that the assessment would be correct, thus the prototype is designed to aid the CAB. Results might be biased up to a certain point due to construction of the examples from Bosch and Fabasoft. Shortcomings in this regard could be mitigated by using same metric on different policy documents, as then either the metric or the extraction method needs to be adapted.

For this proof-of-concept component, no log files or screenshots of evidence will be covered, the focus is on policy documents. Given the structure of the dataset and the focus on building a prototype that works; the organisational metrics are specific to policy documents and no other "raw" evidence sources. Future work could extend the prototype to work with other document types as well as forms of evidence. However as most of other document types used to write policies can be converted to PDF anyway this is not considered for now.

(cc) BY-SA

6 Conclusions

This document presents a technical report about the design, architecture, and current implementation states of MEDINA evidence gathering components. It details the individual components' functions, internal structure and their subcomponents. Information about the current and general limitations and future planned work is also presented.

The components presented in this document include three tools supporting the security assessment of cloud infrastructure (Clouditor, Wazuh, and Vulnerability Assessment Tools), a pair of tools for assessing the security and compliance of cloud application's source code (Codyze and CloudPG), and a component for the assessment of organisational measures based on analysis of CSP's documentation (Assessment and Management of Organisational Evidence, AMOE).

At this point in the project, the described components satisfy most of their functional requirements elicited in scope of WP5 and presented in D5.2 [4]. An overview of the current fulfilment of requirements by the implemented tools is presented in Appendix A. The components are also integrated with the other parts of MEDINA framework in a large extent.

The components, presented in this deliverable, are being continuously integrated into the MEDINA framework in the scope of WP5. The next and final iteration of this deliverable, D3.6 [29], will give a report on the final version of the components in month 30 (April 2023).

(cc) BY-SA

7 References

[1] ENISA, “EUCS – Cloud Services Scheme,” Draft version provided by ENISA (August 2022) - not intended for being used outside the context of MEDINA, 2022.

[2] MEDINA Consortium, “D3.4 Tools and techniques for collecting evidence of tehcnical and organisational measures - v1,” 2021.

[3] MEDINA Consortium, “D3.2 Tools and techniques for the management of trustworthy evidence - v2,” 2022.

[4] MEDINA Consortium, “D5.2 MEDINA requirements, Detailed architecture, DevOps infrastructure and CI/CD and verification strategy - v2,” 2022.

[5] MEDINA Consortium, “D2.4 Specification of the Cloud Security Certification Language - v2,” 2022.

[6] MEDINA Consortium, “D4.2 Tools and techniques for the management and evaluation of cloud security certifications-v2,” 2022.

[7] MEDINA Consortium, “D2.1 Continuously certifiable technical and organizational measures and catalogue of cloud security metrics-v1,” 2021.

[8] Wazuh Inc., “Wazuh,” [Online]. Available: https://wazuh.com/. [Accessed October 2022].

[9] Cisco, “ClamAV,” [Online]. Available: https://www.clamav.net/. [Accessed October 2022].

[10] Chronicle Security, “VirusTotal,” [Online]. Available: https://www.virustotal.com/. [Accessed October 2022].

[11] “w3af,” [Online]. Available: http://w3af.org/. [Accessed September 2022].

[12] OWASP Foundation, “OWASP Zed Attack Proxy (ZAP),” [Online]. Available: https://owasp.org/www-project-zap/. [Accessed September 2022].

[13] “Nmap,” [Online]. Available: https://nmap.org/. [Accessed September 2022].

[14] Docker, Inc., “Docker,” [Online]. Available: https://www.docker.com/. [Accessed October 2022].

[15] Rapid7, “Metasploit,” [Online]. Available: https://www.metasploit.com/. [Accessed October 2022].

[16] VMware, Inc., “RabbitMQ,” [Online]. Available: https://www.rabbitmq.com/. [Accessed October 2022].

[17] MongoDB, Inc., “MongoDB,” [Online]. Available: https://www.mongodb.com/. [Accessed October 2022].

CC BY-SA

[18] OpenStack, "OpenStack Swift (Github repository)," [Online]. Available: https://github.com/openstack/swift. [Accessed October 2022].

[19] Google LLC, "Angular," [Online]. Available: https://angular.io/. [Accessed October 2022].

[20] Faraday Security, "Faraday (Github repository)," [Online]. Available: https://github.com/infobyte/faraday. [Accessed October 2022].

[21] HashiCorp, Inc., "Vagrant," [Online]. Available: https://www.vagrantup.com/. [Accessed October 2022].

[22] Red Hat, Inc., "Ansible," [Online]. Available: https://www.ansible.com/. [Accessed October 2022].

[23] CYBERWISER.eu consortium, "CYBERWISER.eu," [Online]. Available: https://www.cyberwiser.eu/. [Accessed October 2022].

[24] M. Deng, K. Wuyts, R. Scandariato, B. Preneel and W. Joosen, "A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements," *Requirements Engineering,* vol. 16, pp. 3-32, 2011.

[25] C. Banse, I. Kunz, A. Schneider and K. Weiss, "Cloud Property Graph: Connecting Cloud Security Assessments with Static Code Analysis," in *14th IEEE International Conference on Cloud Computing (CLOUD)*, 2021.

[26] Fraunhofer AISEC, "MARK (Modeling Language for Cryptography Requirements and Guidelines) GitHub page," [Online]. Available: https://github.com/Fraunhofer-AISEC/codyze-mark-eclipse-plugin. [Accessed October 2022].

[27] OASIS SARIF TC, "Static Analysis Results Interchange Format (SARIF) Version 2.1.0," 27 March 2020. [Online]. Available: https://docs.oasis-open.org/sarif/sarif/v2.1.0/os/sarif-v2.1.0-os.html.

[28] F. Yamaguchi, N. Golde, D. Arp and K. Rieck, "Modeling and Discovering Vulnerabilities with Code Property Graphs," in *2014 IEEE Symposium on Security and Privacy*.

[29] MEDINA Consortium, "D3.6 Tools and techniques for collecting evidence of technical and organisational measures-v3," 2023.

(cc) BY-SA

Appendix A. MEDINA requirements implementation overview

Table 9 below presents an overview of requirements and their fulfilment with the currently implemented tools presented in this document. The requirements were elicited in WP5 and are detailed in D5.2 [4]. For the common requirements, implementation status is given for all the related components, while tool-specific requirements are presented in groups according to their respective components, such as they are also structured in D5.2. Implementation status has three possible values represented in the table by colours:

- Green: fully implemented
- Orange: partially implemented
- Red: not implemented

This table is updated and presented in all versions of this report for easier comparison and progress tracking. Note that some requirements were added, moved between categories, and/or their titles changed since the initial version of this deliverable.

Table 9. Overview of requirements satisfaction according to current implementation of presented tools

Requirement ID	Short title	Implementation status			
Common requirements for technical evidence gathering		***Clouditor***	***Wazuh***	***VAT***	***Codyze***
TEGT.C.01	Continuous collection				
TEGT.C.02	Provision to defined interfaces				
Clouditor (Gathering evidence from cloud interfaces)					
TEGT.S.01	Collect evidence from cloud interfaces				
EAT.02	Continuous evidence assessment				
Clouditor (Security assessment)					
EAT.01	Evidence assessment target				
EAT.03	Evidence assessment results				
Clouditor (Evidence orchestration)					
ECO.01	Provision of Interfaces				
ECO.02	Conformity to selected assurance level				
ECO.03	Secure Transmission to evidence storage				
ECO.04	Transmission of evidence checksums				
Clouditor (Gathering evidence from CSP-Native Services)					
TEGT.S.09	Collect evidence from CSP-native services				
EAT.04	Assess CSP-Native Evidence				
Wazuh (Gathering evidence from computing resources)					
TEGT.S.08	Provision of malware, intrusion, and vulnerability detection tools				
VAT (Gathering evidence from computing resources)					
TEGT.S.08	Provision of malware, intrusion, and vulnerability detection tools				
CloudPG (Gathering evidence from application source code)					
TEGT.S.02	Collect evidence from source code via CPG				

CC BY-SA

TEGT.S.03	Implement information and data flow analysis	
TEGT.S.10	Connect infrastructure- and application-level security analyses	
TEGT.S.07	Support for common programming languages, libraries, cloud services	
Codyze (Gathering evidence from application source code)		
TEGT.S.03	Implement information and data flow analysis	
TEGT.S.04	Support expression of security requirements	
TEGT.S.05	Verify security requirements	
TEGT.S.06	Retrieve source code of cloud applications	
TEGT.S.07	Support for common programming languages, libraries, cloud services	
TEGT.S.08	Provision of malware, intrusion, and vulnerability detection tools	
Assessment and Management of Organisational Evidence (AMOE)		
OEGM.01	Continuous collection of organizational evidence	
OEGM.02	Provision to defined interfaces	
OEGM.03	Usability for auditors	
OEGM.04	Minimum evidence storage	
OEGM.05	Evidence Assessment results	

In total, there are 35 requirements (if the common requirements are counted separately – once for each component) related to the presented components. 24 (68%) of them are currently marked as fully implemented, 9 (26%) as partly implemented, and 2 (6%) as not implemented. The basic statistic of requirement coverage by each component is presented in Table 10.

Table 10. Requirements satisfied by each tool

Tool	Number of requirements	Fully implemented	Partially implemented	Not implemented
Clouditor	12	9	1	2
Wazuh	3	2	1	0
VAT	3	2	1	0
CloudPG	4	4	0	0
Codyze	8	2	6	0
AMOE	5	5	0	0

(cc) BY-SA

Appendix B. Clouditor README, installation instructions and user manual

1. README

Clouditor Community Edition

Introduction

Clouditor is a tool which supports continuous cloud assurance. Its main goal is to continuously evaluate if a cloud-based application (built using, e.g., Amazon Web Services (AWS) or Microsoft Azure) is configured in a secure way and thus complies with security requirements defined by, e.g., Cloud Computing Compliance Controls Catalogue (C5) issued by the German Office for Information Security (BSI) or the Cloud Control Matrix (CCM) published by the Cloud Security Alliance (CSA).

Features

Clouditor currently supports over 60 checks for Amazon Web Services (AWS), Microsoft Azure and OpenStack. Results of these checks are evaluated against security requirements of the BSI C5 and CSA CCM.

Key features are:

- automated compliance rules for AWS and MS Azure,
- granular report of detected non-compliant configurations,
- quick and adaptive integration with existing service through automated service discovery,
- descriptive development of custom rules using Cloud Compliance Language (CCL) to support individual evaluation scenarios,
- integration of custom security requirements and mapping to rules.

Build

Install necessary protobuf tools.

```
go install google.golang.org/protobuf/cmd/protoc-gen-go \
google.golang.org/grpc/cmd/protoc-gen-go-grpc \
github.com/grpc-ecosystem/grpc-gateway/v2/protoc-gen-grpc-gateway \
github.com/google/gnostic/cmd/protoc-gen-openapi
```

Also make sure that `$HOME/go/bin` is on your `$PATH` and build:

```
go generate ./...
go build -o ./engine cmd/engine/engine.go
```

Usage

To test, start the engine with an in-memory DB

```
./engine --db-in-memory
```

Alternatively, be sure to start a postgre DB:

```
docker run -e POSTGRES_HOST_AUTH_METHOD=trust -d -p 5432:5432 postgres
```


(cc) BY-SA

Clouditor CLI

The Go components contain a basic CLI command called `cl`. It can be installed using `go install cmd/cli/cl.go`. Make sure that your `~/go/bin` is within your $PATH. Afterwards the binary can be used to connect to a Clouditor instance.

```
cl login <host:grpcPort>
```

Command Completion

The CLI offers command completion for most shells using the `cl completion` command. Specific instructions to install the shell completions can be accessed using `cl completion --help`.

2. Installation instructions

The full up-to-date installation instructions can be found in the README at the Clouditor Github repository[2].

To build Clouditor, the Gradle build tool[26] is used. To enable an auto-discovery for AWS and/or Azure the credentials must be stored in the home folder.

Since *Protobuf* is used, the corresponding packages must also be installed (the installation command can be found in the README):

- *google.golang.org/protobuf/cmd/protoc-gen-go*
- *google.golang.org/grpc/cmd/protoc-gen-go-grpc*
- *github.com/grpc-ecosystem/grpc-gateway/v2/protoc-gen-grpc-gateway*
- *github.com/googleapis/gnostic/apps/protoc-gen-openapi*

The Clouditor features its own CLI for which *~/go/bin* must be within the *$PATH environment variable.*

To build the prototype make sure that *$HOME/go/bin* is within your *$PATH* and run the following commands:

- *go generate ./...*
- *go build ./…*

The engine can be started by using an in-memory DB as well as a Postgres DB. To start the engine with an in-memory DB, use *./engine –db-in-memory* if starting with a separate Postgres DB use *./engine*. Start the Postgres DB.

For development, an overview for the installation instructions is given in the following:

- Build Clouditor with Gradle or alternatively via a docker image
- Build Go components (*Protobuf* tools needed for compiling the *Protobuf* files)
- Start the Clouditor with in-memory DB or a Postgres DB
- Install and use the CLI for running the Clouditor at runtime

[26] https://github.com/clouditor/clouditor/blob/main/README.md

CC BY-SA

3. User Manual

The Clouditor components can be used with CLI commands. The help is shown by running *cl –help:*

```
user@user:~$ cl --help
The Clouditor CLI

Usage:
  cl [command]

Available Commands:
assessment-result      Assessment result commands
cloud                  Target cloud services commands
completion             Generate completion script
evidence               Evidence commands
help                   Help about any command
login                  Log in to Clouditor
metric                 Metric commands
requirement            Requirement commands
resource               Resource commands
service                Service commands
tool                   Tool commands

Flags:
  -h, --help                        help for cl
  -s, --session-directory string    the directory where the session will be saved and loaded from (default "/home/user/.clouditor/")

Use "cl [command] --help" for more information about a command.
```

Each command can have additional subcommands which are explained by the corresponding help, e.g., *cl assessment-result –help*.

Please note that before using the Clouditor CLI it is necessary to login to Clouditor: *cl login <host:grpcPort>.*

(cc) BY-SA

Appendix C. Codyze installation instructions and user manual

1. Installation instructions

The latest installation instruction for MEDINA Codyze are described in the README available in the public MEDINA repository[27]. In addition, the latest installation instructions for each component are available in their respective GitHub repositories and on the main Codyze website – https://www.codyze.com/.

Codyze for MEDINA is built with Gradle. The project repository contains the Gradle wrapper. To build Codyze for MEDINA two build steps are required. First, the REST API for the Orchestrator needs to be built. It is generated from the OpenAPI specifications distributed by the Orchestrator. The build commands are:

- `./gradlew[.bat] generateAll`
- `./gradlew[.bat] installDist`

After the build, the Codyze for MEDINA executable is located at {project-dir}/build/install/codyze/. In this directory one can find three directories:

- bin/ contains a shell Windows batch script to run Codyze
- lib/ contains all library files
- mark/ contains the MARK files included in Codyze

The start scripts in bin/ will print a command help when executed. The command help contains short descriptions of each command argument and parameter.

The components used by Codyze for MEDINA are consumed as library dependency and automatically retrieved when Codyze for MEDINA is built from source. In case the components need to be built from source, their respective code repositories contain up to date information on the build instruction, prerequisites, and procedure.

Finally, the MARK plugin for Eclipse IDE is provided by an Eclipse update site. The installation of this plugin is described on the Codyze website.

2. User Manual

The user manual for MEDINA Codyze is available as README in the public MEDINA repository[27]. In addition, the Codyze library and MARK are documented at the Codyze website – https://www.codyze.io/ -- and at their respective GitHub repositories.

[27] https://git.code.tecnalia.com/medina/public/codyze

CC BY-SA

Appendix D. Cloud Property Graph installation instructions and user manual

1. Installation

Note that the installation instructions may change with the advancement of the tool, so consider the installation details in the README file on the GitHub repository[28]. The following instructions and the following manual are partly copied from this file:

1. Clone the git repository git@github.com:clouditor/cloud-property-graph.git
2. Set the JAVA_HOME variable to Java 11
3. Install jep, follow the instructions at https://github.com/Fraunhofer-AISEC/cpg#python
4. For usage of experimental language, e.g., go
    a. Checkout Fraunhofer AISEC - Code Property Graph and build by using the property -Pexperimental: ./gradlew build -Pexperimental
    b. The libcpgo.so must be placed somewhere in the java.library.path. (For further informaton see https://github.com/Fraunhofer-AISEC/cpg#usage-of-experimental-languages)
        i. Under Linux in /lib/. sudo cp ./cpg-library/src/main/golang/libcpgo.so /lib/
        ii. And Mac in ~/Library/Java/Extensions.
5. To build, the graph classes need to be built from the Ontology definitions by calling ./build-ontology.sh. Then build using ./gradlew installDist.

2. User Manual

Start neo4j using *docker run -d --env NEO4J_AUTH=neo4j/password -p7474:7474 -p7687:7687 neo4j* or *docker run -d --env NEO4J_AUTH=neo4j/password -p7474:7474 -p7687:7687 neo4j/neo4j-arm64-experimental:4.3.2-arm64* on ARM systems.

Run *cloudpg/build/install/cloudpg/bin/cloudpg*. This will print a help message with any additional needed parameters. The root path is required, and the program can be called as follows: *cloudpg/build/install/cloudpg/bin/cloudpg --root=/x/testprogramm folder1/ folder2/ folder 3/*

[28] https://github.com/clouditor/cloud-property-graph/blob/main/README.md

CC BY-SA

Appendix E. AMOE user manual

To use the AMOE GUI start by clicking on the "Upload new file" button. Then a file upload dialog is shown as depicted in Figure 13.

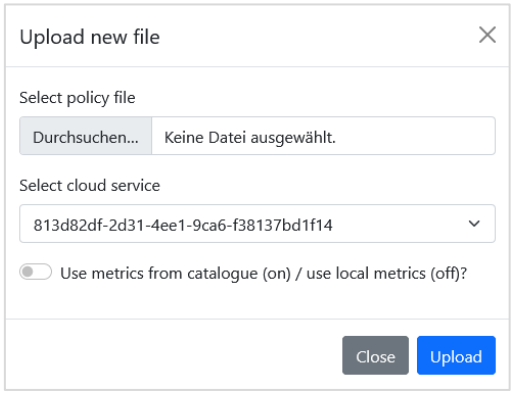

Figure 13. AMOE file upload dialog

Select a policy PDF document for uploading and the cloud service (id) that should be connected to. Then click Upload.

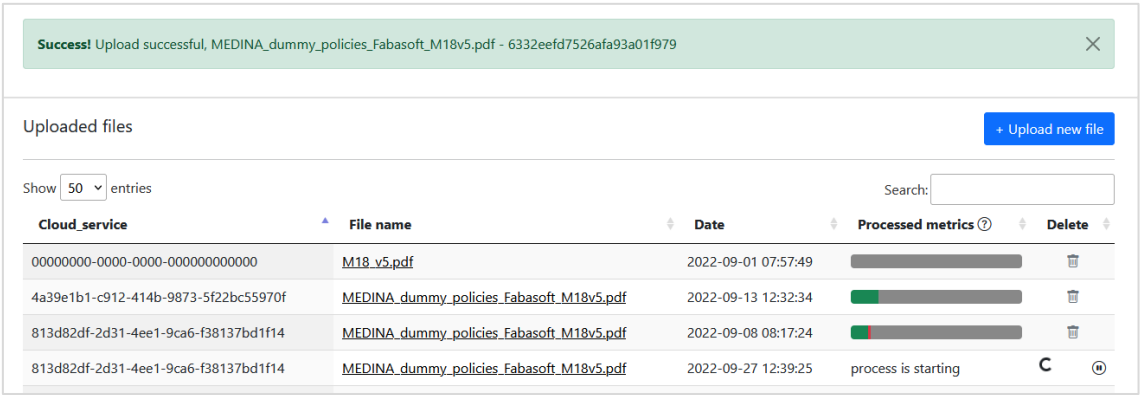

Figure 14. AMOE landing page after file upload

The evidence extraction process is started in the background. It can take some time until every organisational metric has been processed. The process can be stopped by clicking on ⏸. Files and their linked evidence can be deleted by clicking on 🗑.

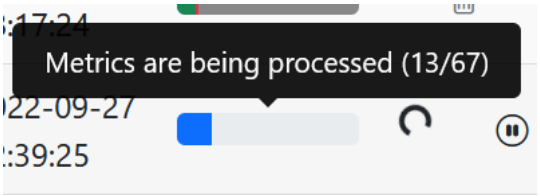

Figure 15. AMOE evidence extraction progress

Figure 15 depicts the progress of the background evidence extraction process. On hovering, details are shown.

(cc) BY-SA

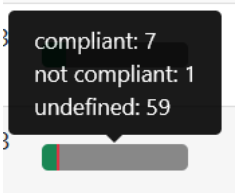

Figure 16. AMOE assessment status overview per document

In Figure 16 the status overview is shown. This is displayed, for every file after the evidence extraction process has finished. The details are shown by hovering with the mouse. Green indicates the number of assessment results set to compliant, red the number set to not compliant and grey marks where no status has been set (undefined).

To view the evidence results of an uploaded file, click on a row of the respective table or the filename of the list as depicted in Figure 14. The overview as depicted in Figure 17 opens. The overview contains meta data of the uploaded file, filter, and search options. In the case an assessment result has been set, it can be submitted to the Orchestrator directly from this view. Otherwise, click on a row to get to the detailed view for the extracted evidence.

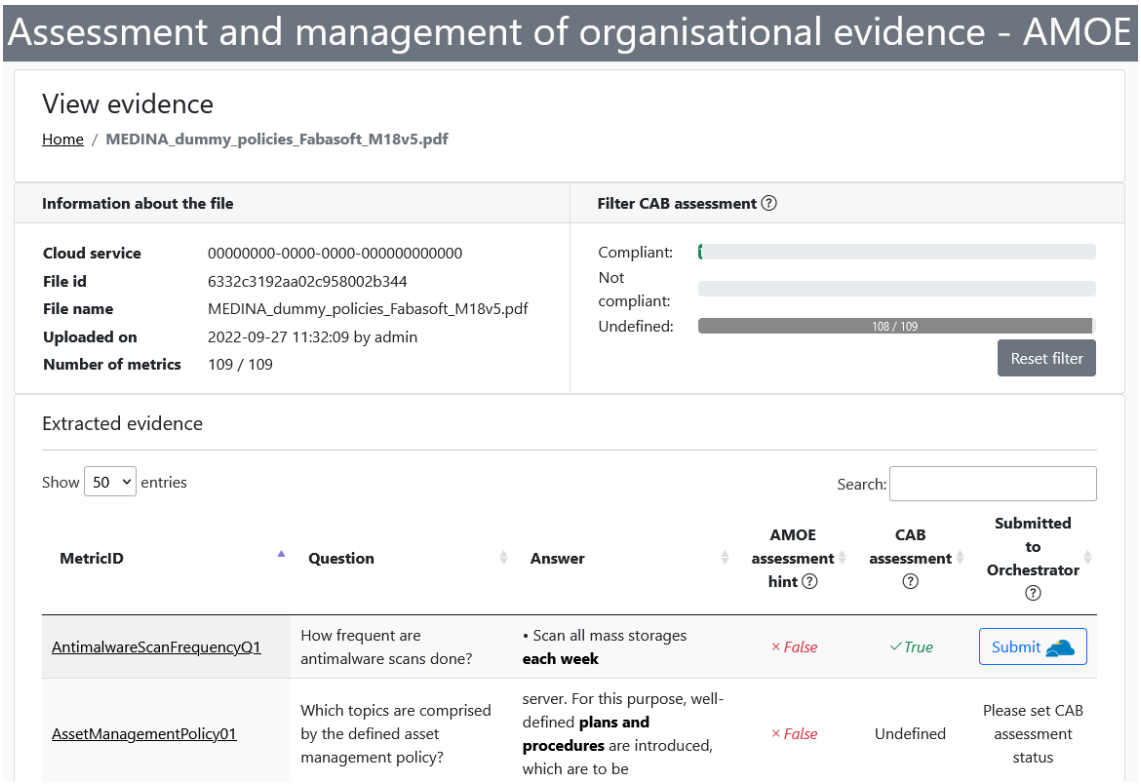

Figure 17. AMOE overview of extracted evidence and meta data linked to the uploaded file

Figure 18 depicts the detailed view of the extracted evidence. The linked requirement is shown on the top. This is followed by the metric meta data, extracted answer, assessment hint and options to set the assessment result and comment. At the bottom of the page, the processed HTML version of the document is shown. The extracted evidence is highlighted in green.

CC BY-SA

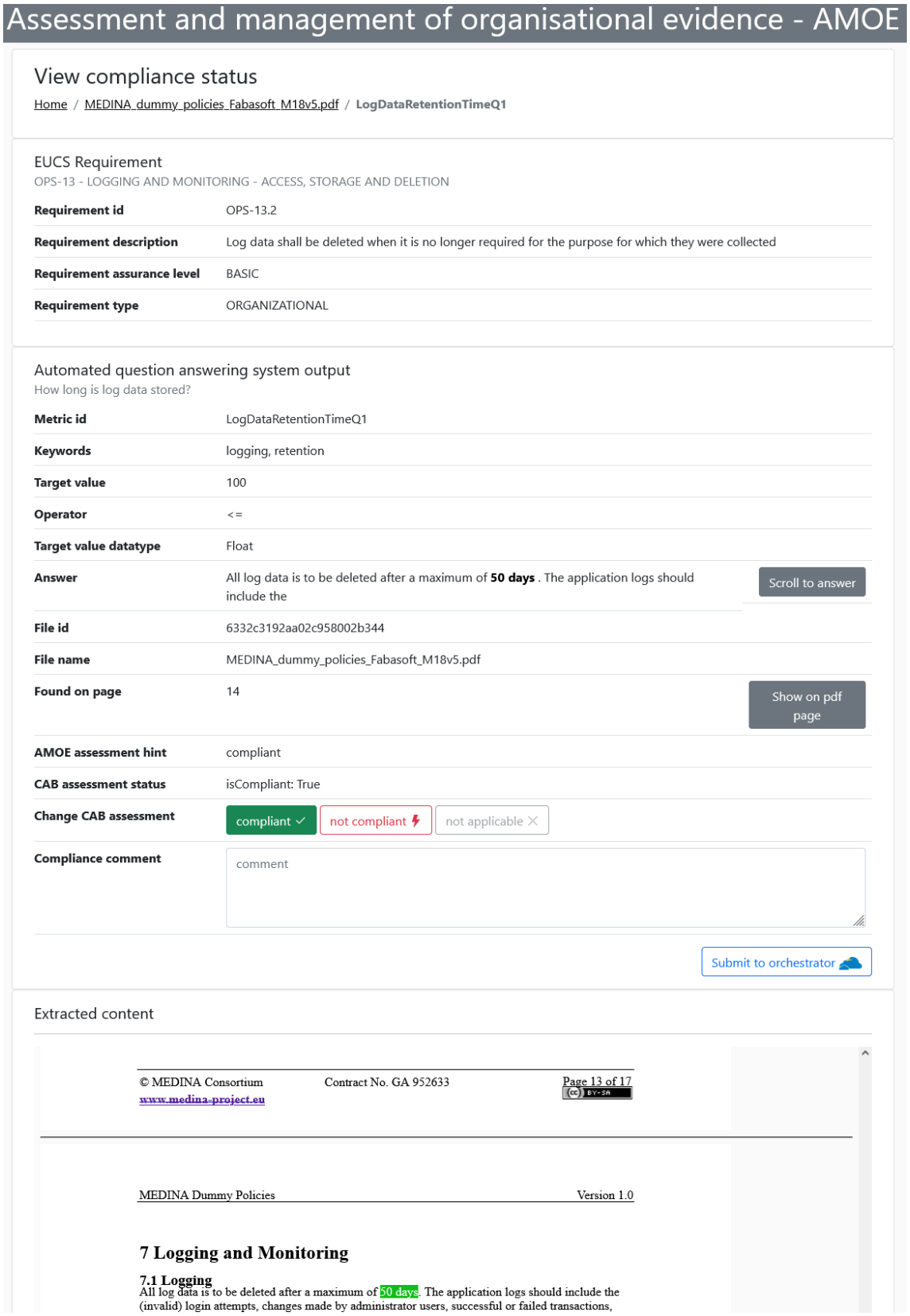

Figure 18. AMOE view of organisational evidence