EMERALD

Deliverable D2.2

Source Evidence Extractor – v1

Editor(s):	Verena Geist, Stefan Schöberl
Responsible Partner:	Software Competence Center Hagenberg GmbH (SCCH)
Status-Version:	Final – v1.0
Date:	31.10.2024
Type:	OTHER
Distribution level:	PU

Project Number:	101120688
Project Title:	EMERALD

Title of Deliverable:	D2.2 – Source Evidence Extractor – v1
Due Date of Delivery to the EC	31.10.2024

Workpackage responsible for the Deliverable:	WP2 – Methodology for knowledge extraction
Editor(s):	Verena Geist, Stefan Schöberl (SCCH)
Contributor(s):	Florian Wendland (FHG)
Reviewer(s):	Ramon Martin De Pozuelo Genis (CXB) Juncal Alonso (TECNALIA) Cristina Martinez (TECNALIA)
Approved by:	All Partners
Recommended/mandatory readers:	WP1, WP2, WP3, WP4, and WP5

Abstract:	This deliverable presents tools and techniques for evidence extraction from source code that can be integrated with the certification graph. It is the result of work performed in Task 2.2. This document is a first/interim version, the final version on source evidence extractors will be reported in D2.3.
Keyword List:	Knowledge extraction, source code files, technical evidence, static code analysis, Codyze, eknows.
Licensing information:	This work is licensed under Creative Commons Attribution-ShareAlike 4.0 International (**CC BY-SA 4.0 DEED** https://creativecommons.org/licenses/by-sa/4.0/)
Disclaimer	Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. The European Union cannot be held responsible for them.

Document Description

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	14.08.2024	First draft version, ToC	Verena Geist, Stefan Schöberl (SCCH), Florian Wendland (FHG)
v0.2	22.08.2024	Executive summary	Verena Geist (SCCH)
v0.3	26.08.2024	References, introduction, embedding into the EMERALD architecture, acronyms	Verena Geist (SCCH)
v0.4	04.09.2024	Functional description of *eknows*	Verena Geist (SCCH)
v0.5	25.09.2024	Descriptions of *Codyze*	Florian Wendland (FHG)
v0.6	01.10.2024	Technical description of *eknows*	Stefan Schöberl (SCCH)
v0.7	07.10.2024	Conclusion, finalizing the document for the internal review	Verena Geist (SCCH)
v0.8	17.10.2024	Internal review	Ramon Martin De Pozuelo Genis (CXB)
v0.9	18.10.2024	Revision of internal review	Verena Geist, Stefan Schöberl (SCCH)
v0.10	28.10.2024	Final reviewed version	Juncal Alonso, Cristina Martinez (TECNALIA)
v1.0	31.10.2024	Submitted to the European Commission	Juncal Alonso, Cristina Martínez (TECNALIA)

Table of contents

Terms and abbreviations 6
Executive Summary 8
1 Introduction 9
1.1 About this deliverable 9
1.2 Document structure 10
2 Source evidence extractors in the EMERALD architecture 11
3 Codyze for EMERALD 12
3.1 Functional description 12
3.2 Technical description 13
3.2.1 Prototype architecture 13
3.2.2 Technical specifications 15
3.3 Delivery and usage 16
3.3.1 Package information 16
3.3.2 Installation 16
3.3.3 Instructions for use 16
3.3.4 Licensing information 17
3.3.5 Download 17
3.4 Limitations and future work 17
4 eknows evidence extractor 18
4.1 Functional description 18
4.2 Technical description 22
4.2.1 Prototype architecture 22
4.2.2 Technical specifications 23
4.3 Delivery and usage 24
4.3.1 Package information 24
4.3.2 Installation 24
4.3.3 Instructions for use 25
4.3.4 Licensing information 25
4.3.5 Download 25
4.4 Limitations and future work 25
5 Conclusions 27
6 References 28
Appendix A: eknows Binary Usage Software License 30

List of tables

TABLE 1. REQUIREMENT CODYZE.01 - EXTRACTION OF SECURITY FEATURES FROM SOURCE CODE 13
TABLE 2. SUPPORTED PROGRAMMING LANGUAGES BY CODYZE THROUGH THE CPG LIBRARY 15
TABLE 3. OVERVIEW OF CODYZE PACKAGE STRUCTURE 16
TABLE 4. EKNOWS.01 - INTEGRATION INTO EXISTING SYSTEMS 19
TABLE 5. EKNOWS.02 - RESILIENCE WHILE ANALYSING ERRONEOUS CODE 19
TABLE 6. EKNOWS.03 - MULTI-LANGUAGE SUPPORT 20
TABLE 7. EKNOWS.04 - SUPPORT EMERALD EVIDENCE FORMAT 20
TABLE 8. EKNOWS.05 - STATIC CODE ANALYSIS 21
TABLE 9. SUPPORTED PROGRAMMING LANGUAGES BY EKNOWS 23
TABLE 10. OVERVIEW AND DESCRIPTION OF PACKAGE STRUCTURE FOR THE EKNOWS EVIDENCE EXTRACTOR 24

List of figures

FIGURE 1. EMERALD COMPONENT OVERVIEW DIAGRAM [9]. THE RED RECTANGLE HIGHLIGHTS THE SOURCE EVIDENCE EXTRACTION COMPONENTS, WHICH ARE DESCRIBED IN THIS DELIVERABLE. 11
FIGURE 2. ARCHITECTURE OF CODYZE FOR EMERALD HIGHLIGHTING ITS MODULES AND CONTRIBUTIONS WITHIN THE EMERALD PROJECT (I.E., MODULES WITH DASHED BOXES ARE EXTERNAL) 14
FIGURE 3. CONFIGURATION OF EVIDENCE COLLECTORS IN THE EMERALD UI (D4.3 [14]) 17
FIGURE 4. REVERSE ENGINEERING ACTIVITIES SUPPORTED BY THE SOFTWARE PLATFORM EKNOWS [15]. FURTHER EXPLANATIONS OF SUBCOMPONENTS ARE PROVIDED IN SECTION 4.2.1.1. 18
FIGURE 5. OVERVIEW OF EKNOWS PLATFORM BUILDING BLOCKS [16] 22

Terms and abbreviations

AI	Artificial Intelligence
AI-SEC	AI Security Evidence Collector
AMOE	Assessment and Management of Organisational Evidence
ANTLR	ANother Tool for Language Recognition
API	Application Programming Interface
AST	Abstract Syntax Tree
ASTM	Abstract Syntax Tree Metamodel
BSI C5	BSI Cloud Computing Compliance Criteria Catalogue
CertGraph	Certification Graph
CaaS	Container-as-a-Service
CDT	C/C++ Development Tooling
CI/CD	Continuous Integration / Continuous Deployment
CIL	Common Intermediate Language
CLI	Command-Line Interface
COBOL	Common Business-Oriented Language
CoCo/R	Compiler Generator
Codyze	Static Code Analyzer from FHG
CPG	Code Property Graph
CSA or EU CSA	EU Cybersecurity Act
CSP	Cloud Service Provider
DoA	Description of Action
DOT	Markup-Language
DSL	Domain-Specific Language
DSM	Domain-Specific Model
EC	European Commission
eknows	The platform for software analysis from SCCH
eknows core	Selected modules of the *eknows* platform, which form the basis for the *eknows* evidence extractor
eknows evidence extractor	The extractor component developed in the context of EMERALD
ENISA	European Union Agency for Cybersecurity
EUCS	EU Cloud Certification Scheme
GA	Grant Agreement to the project
GASTM	Generic AST Metamodel
HTML	Hypertext Markup Language
JAR	Java Archive
JCL	Job Control Language
JDT	Java Development Tools
JNA	Java Native Access
JSON	JavaScript Object Notation
Koopa	(COBOL) Parser Generator
KPI	Key Performance Indicator
KR	Key Result
MD	Markdown
MEDINA	Predecessor project of EMERALD
ODF	Open Document Format
OMG	Object Management Group
PL/I	Programming Language One

PL/SQL	Procedural Language/Structured Query Language
PT	Parse Tree
REST	Representational State Transfer
SARIF	Static Analysis Results Interchange Format
SASTM	Specialized AST Metamodel
SE	Standard Edition
SVG	Scalable Vector Graphics
SW	Software
TLS	Transport Layer Security
TRL	Technology Readiness Level
WP	Work Package

Executive Summary

This deliverable presents the initial design, architecture, and implementation state of the source evidence extractors of WP2, i.e., *Codyze* and *eknows evidence extractor*. They contribute to the key result KR1-EXTRACT of EMERALD, a framework to continuously extract knowledge from different layers of a cloud service and prepare suitable evidence based on them.

EMERALD follows a knowledge graph-based approach to provide a unified view of the cloud service under certification at different layers of the service, ranging from the infrastructure layer (e.g., virtual resources), to the business layer (e.g., policies and procedures), to the implementation layer (e.g., source code files) and data layer (e.g., increasingly used AI models) in cloud applications. The source evidence extractors, developed in Task 2.2 and described in this deliverable, aim at identifying critical security-related functionality such as data encryption, transport encryption, or authentication in source code. Other related deliverables in WP2, all due at project month 12 (October 2024), provide functional and technical details on further evidence extractors from different sources, i.e., D2.4 [1] on evidence extraction from policy documents in Task 2.3, D2.6 [2] on security and privacy preserving evidence extraction in Task 2.4, and D2.8 [3] on runtime data extraction in Task 2.5. All these details contributed to D2.1 [4] on the overall information model of the certification graph in Task 2.1.

This document starts by illustrating how the source evidence extractors fit into the overall EMERALD architecture. The main part provides functional and technical descriptions of the two extractor components *Codyze* and *eknows evidence extractor*, including their purpose and scope, the (current and planned) coverage of the EMERALD requirements, the components' internal architecture and their subcomponents. These descriptions are complemented by information on delivery and usage, as well as on limitations and future work. Finally, the document concludes with a short summary.

The source evidence extractors described in this deliverable contribute to KR1-EXTRACT by providing next-generation evidence gathering tools and techniques based on a knowledge graph approach. The presented extractors currently have the initial prototypes implemented and ready to be (to some degree) integrated with other components of the EMERALD architecture. Some requirements of the components are already fully or partially satisfied by the presented prototypes.

Based on the work described in this deliverable, the source evidence extractors will be further extended and integrated into the EMERALD framework. This is the first iteration of the deliverable coming from Task 2.2. The second and final version of this deliverable with the updated extractors will be delivered with D2.3 [5] in project month 24 (October 2025). Evidence will be prepared according to the integrated, graph-based model of semantically linked and combined evidence, provided in D2.10 (interim version) [6] in project month 15 (January 2025) and D2.11 (final version) [7] in project month 27 (January 2026). The extracted evidence will be stored and assessed, i.e., to verify the implementation of security metrics, in the scope of WP3.

1 Introduction

EMERALD aims to provide a next generation set of evidence gathering tools and techniques based on a knowledge graph approach. KR1-EXTRACT supports an improved and unified tool-supported approach to continuously extract knowledge from different layers of a cloud service, e.g., infrastructure, platform, runtime information, policy documents, software, and AI models.

The objective of WP2 is to establish a unified view of the cloud service under certification by extracting and enriching knowledge of the different layers of the service and providing suitable evidence for security metrics. A major part of this work package is research and design of multiple tools and techniques to extract knowledge out of various sources. A graph-based model, called the certification graph (*CertGraph*), serves as a common structure that is filled by all evidence extraction tools.

1.1 About this deliverable

The goal of this deliverable is to present the design and implementation of the EMERALD evidence extractors, that extract knowledge from source code. This is a report on the initial prototypes reflecting an early stage of implementation and integration of these extractors and is the first of two iterations of deliverables, resulting from Task 2.2.

Evidence on the source code level is primarily gathered by the source evidence extractors *Codyze* and *eknows evidence extractor*, which are adapted to support the *CertGraph* data model. *Codyze*, originally launched in MEDINA[1], focuses on generating evidence for security-related findings, such as the existence of encryption or proper authentication. In EMERALD, it should be advanced to TRL 7 and improved to verify that functionality is implemented according to state-of-the art security guidelines and standards. To supplement evidence extraction from source code, the software analysis platform *eknows is* integrated as basis for the *eknows evidence extractor*. *eknows* offers language-independent analyses and could be extended to identify security-enforcing business rules in code and verify correct usage of security-related APIs. A compact overview of both source code extractors in the form of *Components Cards* can be found in D1.3 [8].

Furthermore, information from project configurations and deployment files such as *infrastructure-as-code* might be used to further augment the *CertGraph* data model. For now, please note that these are ideas of what technical evidence could be gathered from source code. The next deliverable, D2.3 [5], will describe how evidence extraction was implemented in detail according to the agreed security scheme(s). Also note that the integration of *Codyze* and *eknows* is not planned. The EMERALD framework works with two different source evidence extractors, i.e., a security metric may work well with *Codyze*, and another using the *eknows evidence extractor*. However, the resulting evidence format must be the same. This shows the use of APIs in the framework and emphasises that the framework is not tied to a specific tool.

All extracted information together provides a system-level view of the cloud service identifying exposed functionality and interactions with other cloud services. Along these interfaces additional evidence can be gathered specifically for security requirements on service interactions such as transport encryption or authentication. The functionalities are then classified, annotated, and linked with other extracted evidence information from different layers of the cloud service (i.e., infrastructure, policy documents, and artificial intelligence (AI) models) in the EMERALD *CertGraph* [6] [7].

[1] https://medina-project.eu/

1.2 Document structure

The document is structured as follows.

In Section 2, we discuss how the source evidence extractors fit into the overall EMERALD architecture and how they relate to other components.

The main Sections 3 and 4 report on the design and implementation of *Codyze* and the *eknows evidence extractor*. For each source extractor, functional and technical descriptions are provided, including their purpose and scope, the (current and planned) coverage of the EMERALD requirements, the components' internal architecture, their subcomponents, and details about the programming language, libraries, etc. used. These descriptions are complemented by information on delivery and usage, including package information, installation instructions, user manual, licensing and download information, as well as limitations and future work.

Section 5 ends up with the conclusion of this deliverable.

The document includes an appendix, *Appendix A: eknows Binary Usage Software License*, that contains the license for the supplied binaries (*eknows core*, selected parts of the closed source *eknows* platform) which are required by the *eknows evidence extractor*.

2 Source evidence extractors in the EMERALD architecture

This section describes how the source evidence extractors interact with (selected) EMERALD components on a conceptual level. Figure 1 shows the EMERALD high-level architecture as a component diagram [9]. In EMERALD, a component is defined as "any part of the EMERALD ecosystem that has a specific functionality and can be considered a separate entity with respect to other components" [10]. In contrast, a tool is a "software element that has several disparate functions and therefore can be composed by several components" [10]. Therefore, *Codyze* and *eknows* are referred to as components rather than tools in the context of EMERALD.

The components for collecting evidence about technical and organisational measures, i.e., *AMOE*, *eknows*, *AI-SEC, Clouditor-Discovery*, and *Codyze*, are represented at the bottom part of Figure 1. The source evidence extractor components *Codyze* (see Section 3) and *eknows evidence extractor* (see Section 4), which obtain technical evidence from the analysis of the source code of cloud applications, are highlighted using a thick frame. *AMOE* [1], the component for organisational evidence gathering from MEDINA, analyses various documents and policies of the cloud service provider (CSP) and produces evidence about the CSP's compliance to organisational requirements of the certification framework. *Clouditor-Discovery* [3], also originated from MEDINA, collects evidence about the secure configuration of cloud resources, with a focus on runtime data extraction in EMERALD. *AI-SEC* [2] is a newly developed component in EMERALD and analyses AI models for several key evidence regarding robustness against adversarial attacks, explainability, and fairness.

In MEDINA, some extraction components implemented their own assessment, which causes more maintenance effort if requirements change over time. Thus, centralizing assessment is one of the goals in EMERALD. This is done by delivering exclusively (or as far as possible) raw evidence to the *Evidence Store* [8]. All WP2 extraction components, which extract knowledge from the various layers of a cloud service (i.e., policy documents, source code, cloud interfaces, AI models, etc.), provide (part of) evidence (e.g., for transport encryption), which is then mapped to the EMERALD evidence format using the terms described in the *CertGraph Ontology* [4]. This evidence information is stored in the *Evidence Store* following the defined schema and is used to assess the metrics defined in the *Repository of Controls and Metrics* [11].

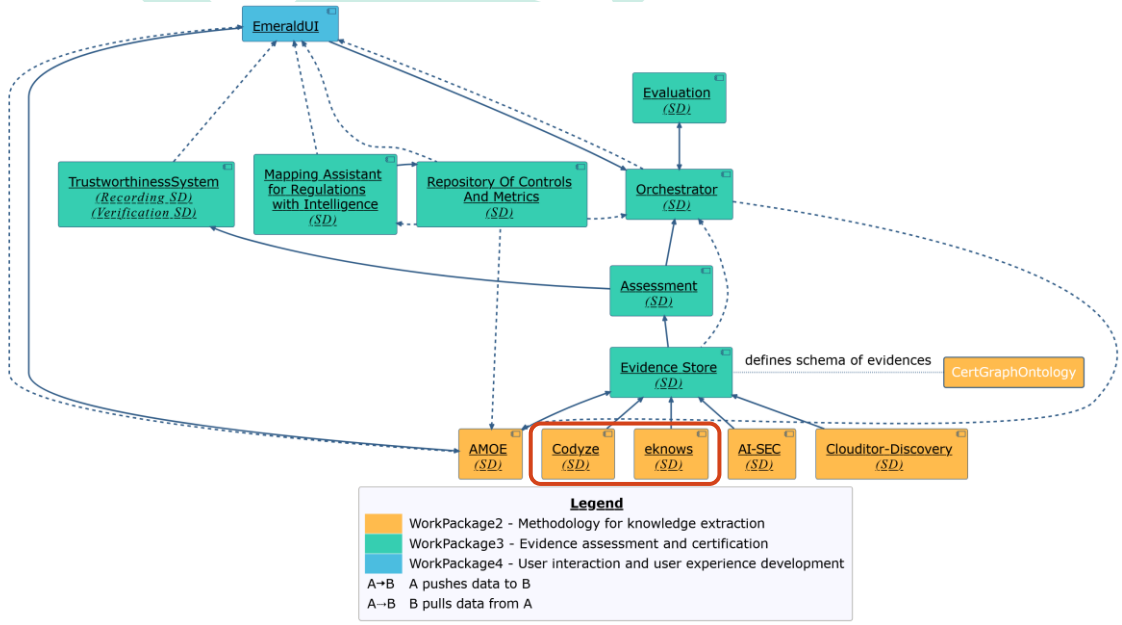

Figure 1. EMERALD component overview diagram [9]. The red rectangle highlights the source evidence extraction components, which are described in this deliverable.

3 Codyze for EMERALD

Codyze[2] is a static code analyser with a focus on verifying compliance in source code. It extracts information from source code and relates it to compliance requirements. Thereby, *Codyze* provides evidence whether an implementation is compliant or non-compliant with respect to specified requirements. Thus, it is possible to perform compliance assessments that incorporate aspects of the software development lifecycle.

3.1 Functional description

Overall purpose. Within the EMERALD framework, *Codyze* provides evidence extraction from source code of cloud services and applications. It identifies code segments that are essential for a good cybersecurity posture and relates them to compliance requirements from certification schemes such as ENISA's EUCS[3]. From the analysis, *Codyze* generates evidence results that indicate if code segments are compliant or non-compliant to specified requirements. These evidence results are submitted to the E*vidence Store* for storage and further processing by the EMERALD framework.

With its analysis, *Codyze* discovers potential compliance violations during software development. This detection enables developers to mend flaws before software is released and deployed. Thus, *Codyze* can reduce the cost of defects and the risk of operating non-compliant cloud services and application. In summary, *Codyze* ensures compliance by design.

Context and scope. *Codyze* for EMERALD is an application that checks software source code for potential compliance violations. Violations are reported such that developers can mend them. To best utilize *Codyze*, it is recommended to run it as part of a CI/CD pipeline as a check. This integration ensures that *Codyze* runs on every code submission and that it prevents the deployment of non-compliant services and application.

Motivation. *Codyze* for EMERALD aims to assist compliance validation in source code during software development. Thus, it provides valuable feedback early in the development lifecycle and to developers, who can mend potential compliance violations in source code. In addition, it prevents the deployment of non-compliant cloud services and applications as a compliance and quality gate during CI/CD.

Requirements. The relevant requirements from D1.3 [12] with their respective implementation state (*partially / fully / not implemented*) and a brief description of how they are / will be implemented are given in Table 1.

Innovation. *Codyze* is going to provide compliance by design through source code analysis. It allows to shift compliance checks left into the software development lifecycle. Historically, code analyses have focused on detecting vulnerabilities. With *Codyze*, analyses results are linked to compliance requirements to provide evidence of compliant software implementations. This evidence is a key part for an overall compliance assessment.

Within EMERALD, *Codyze* is innovated by integrating a common evidence scheme provided by the *CertGraph*. This integration enables an assessment of *Codyze*'s evidence information within the assessment component of the EMERALD framework. It decouples *Codyze*'s specifications for implementations from the compliance assessment. Thus, it enhances the integrability of *Codyze* into the EMERALD framework, improves the usability of *Codyze*'s analysis and supports an extension of use cases covered by *Codyze*.

[2] https://www.codyze.io/
[3] https://www.enisa.europa.eu/publications/eucs-cloud-service-scheme

Table 1. Requirement CODYZE.01 - Extraction of security features from source code

Field	Description
Requirement ID	CODYZE.01
Short title	Extraction of security features from source code
Description	*Codyze* needs to check available source code artefacts for security features.
Status	Work in Progress
Priority	Must
Component	Codyze
Source	Component, KPI
Type	Technical
Related KR	KR1_EXTRACT
Related KPI	KPI 1.1
Validation acceptance criteria	Validated if evidence arrives at the Orchestrator.
Progress	Partially implemented - 20%
Milestone	MS6: Integrated audit suite V2 (M30)
Note	Work depends in parts on and influences KR2_CERTGRAPH

3.2 Technical description

The following subsections describe the technical details of *Codyze* for EMERALD.

3.2.1 Prototype architecture

Codyze for EMERALD consist of an application with CLI. It is executed on the source code of a cloud service or application to be analysed. After the analysis, *Codyze* generates findings pinpointing compliant and non-compliant implementations in the source code. These findings are transformed into evidence specified by the EMERALD framework and classified according to the *CertGraph*'s evidence schema. The evidence information is submitted to the *Evidence Store* for subsequent assessment.

Codyze for EMERALD uses the open source *Codyze library*, which provides the necessary foundation for a specification DSL and evaluators. Moreover, *Codyze library* uses the Code Property Graph library, *CPG library,* to represent source code as a language agnostic code property graph. The *CPG library* implements the source code processing to facilitate the analysis by *Codyze*. Finally, *Codyze* for EMERALD comes with a set of specification files that describe how software implementations relate to compliance requirements. Based on these files, *Codyze* knows what to look for in the source code and how to interpret statements as compliant or non-compliant. Details are represented in Figure 2.

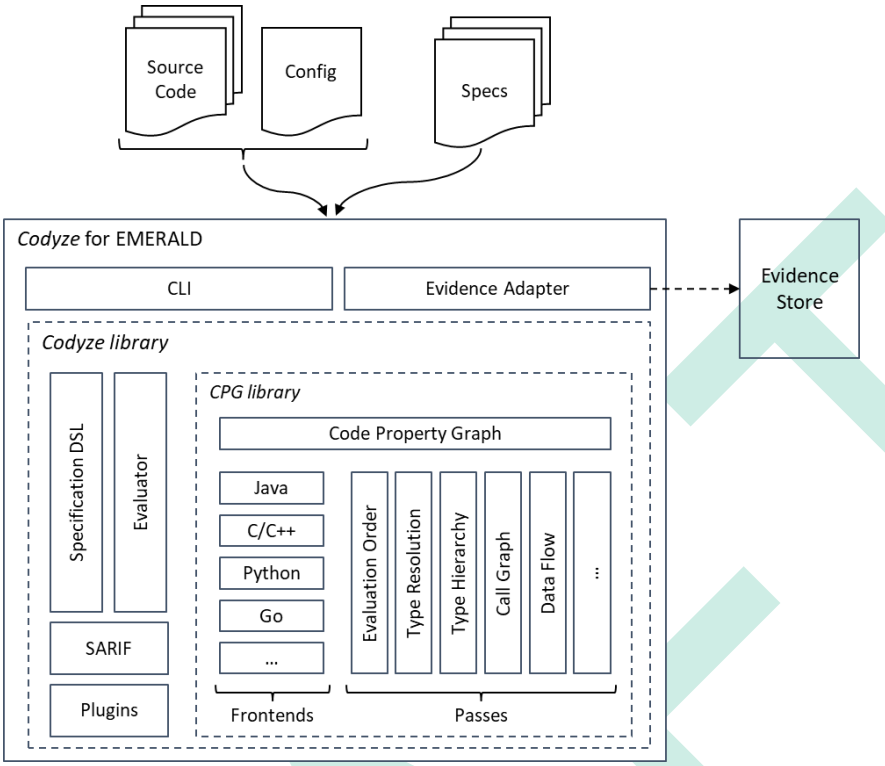

Figure 2. Architecture of Codyze for EMERALD highlighting its modules and contributions within the EMERALD project (i.e., modules with dashed boxes are external)

3.2.1.1 Subcomponents description

CLI. The command-line interface (CLI) is the user's entry point to interact with *Codyze* for EMERALD as an application. It collects the required information such as source code to be analysed, additional configuration options to be applied and specifications to be checked. Based on the provided information, *Codyze* drives the analysis of the source code and generates findings. Through the CLI options, *Codyze* also knows where it needs to submit its findings as evidence. The CLI is designed such that all options and arguments can be provided by a configuration file that can be stored and versioned with the source code of the service or application to be analysed.

Evidence Adapter. The *Evidence Adapter* of Codyze for EMERALD is the main module that integrates *Codyze* into the EMERALD framework. It uses the evidence scheme defined by the *CertGraph* to transform *Codyze*'s findings into EMERALD evidence and submits them to the *Evidence Store*. Thereby, it ensures that findings are correctly mapped to the common evidence types facilitating a compliance assessment within the EMERALD framework.

Codyze Library. The *Codyze library*[4] used by *Codyze* for EMERALD is developed by Fraunhofer AISEC as open-source project under the Apache License, Version 2.0, on GitHub[5]. It defines the domain-specific language (DSL) to specify the connection between software implementations and compliance requirements. In addition, it implements the evaluator for this specification DSL. The evaluator in turn relies on the code representation and functionality of the *CPG library* to evaluate a specified requirement against source code. From the evaluation, the *Codyze library* generates findings indicating compliant and non-compliant parts of source code. These findings are expressed in SARIF[6] – a standard format to describe exchangeable results from static

[4] https://www.codyze.io/
[5] https://github.com/Fraunhofer-AISEC/codyze
[6] https://docs.oasis-open.org/sarif/sarif/v2.1.0/sarif-v2.1.0.html

application analyses, which are further processed by *Codyze* for EMERALD to generate evidence within the EMERALD framework. Moreover, the *Codyze library* provides an extension mechanism through plugins, that can provide additional code analysis capabilities.

The *Codyze library* is only a dependency for *Codyze* for EMERALD. It remains external to the EMERALD project. Necessary contributions for EMERALD will be proposed upstream and code ownership will be transferred.

CPG Library. The *CPG library*[7] used by *Codyze* is developed by Fraunhofer AISEC as an open-source project under the Apache License, Version 2.0, on GitHub[8]. It is an implementation of a code property graph (CPG) [13]. A CPG is a generic graph-based representation of source code enriched with information such as control flow, program dependence, evaluation order, type resolution and call resolution. Thereby, it uses an abstracted representation that allows to represent code from different programming languages with their idiosyncrasies in a language-agnostic format. Currently, the CPG library mainly supports Java, C++, Python and Go. In addition, the abstracted representation supports generic, reusable code exploration and code analyses techniques. The *Codyze library* builds its information extraction and compliance checks on top of the *CPG library*'s exploration and code analysis techniques. Therefore, *Codyze* is limited to analyse code which the *CPG library* supports.

The *CPG library* is only a transitive dependency for *Codyze* for EMERALD through the *Codyze library*. It remains external to the EMERALD project. Necessary contributions for EMERALD will be proposed upstream and code ownership will be transferred.

Specs. *Codyze* defines a domain-specific language (DSL) to allow the specification of compliance requirements as they relate to software implementations. As part of EMERALD, *Codyze* will provide respective specifications to validate the compliance of cloud related compliance schemes such as EUCS.

3.2.2 Technical specifications

Codyze for EMERALD is developed in the programming language Kotlin[9] using a Java Virtual Machine as execution platform. The build system uses Gradle[10] and the project includes the Gradle wrapper to be self-contained. The main libraries of *Codyze* for EMERALD are the *Codyze library* and *CPG library* from GitHub (cf. corresponding sections in 3.2.1.1). As a result of using this *CPG library*, *Codyze* mainly supports the programming languages detailed in Table 2.

Table 2. Supported programming languages by Codyze through the CPG library

Language	CPG module	Parser	Supported version
Java	`cpg-language-java`	JavaParser[11]	Java SE 21
C++	`cpg-language-cxx`	Eclipse CDT[12]	C++17
Python	`cpg-language-python`	Python `ast`[13] (through Jep[14])	Python 3.12
Go	`cpg-language-go`	Go `parser`[15] (through JNA[16])	Go 1.20

[7] https://fraunhofer-aisec.github.io/cpg/
[8] https://github.com/Fraunhofer-AISEC/cpg
[9] https://kotlinlang.org/
[10] https://gradle.org/
[11] https://github.com/javaparser/javaparser
[12] https://projects.eclipse.org/projects/tools.cdt
[13] https://docs.python.org/3/library/ast.html
[14] https://github.com/ninia/jep
[15] https://pkg.go.dev/go/parser
[16] https://github.com/java-native-access/jna

Moreover, *Codyze* produces reports in SARIF. These reports are included in the evidence for the EMERALD framework.

Finally, *Codyze* for EMERALD requires a Java runtime compatible with Java SE 17.

3.3 Delivery and usage

The following subsections detail the delivery and usage of *Codyze* for EMERALD. The provided information is currently work in progress and may change.

3.3.1 Package information

Codyze for EMERALD is delivered in the form of two packages. First, *Codyze* is released as an archive containing all necessary files. The structure of this package is summarized in Table 3. Second, *Codyze* is distributed as a container image. The container image contains the extracted archive of the first package and configures it to be used as an application within a container. Therefore, the installation folder matches the overview of Table 3.

Table 3. Overview of Codyze package structure

Folder / File	Description
`bin/`	Contains execution scripts for Windows and Linux/macOS (POSIX-compliant shells).
`docs/`	Contains detailed documentation texts.
`etc/`	Contains sample configuration files.
`lib/`	Contains application and dependent libraries.
`specs/`	Contains specification files in *Codyze*'s DSL.
`LICENSE`	License text (Apache License, Version 2.0).
`README.md`	Short documentation including short summary description, installation and usage instructions, and further information.

3.3.2 Installation

Installation instructions are provided as part of the README with *Codyze* for EMERALD [18].

In summary, *Codyze* has the following pre-installation requirements:

- Java SE 17 JDK[17].
- Source code of the *Codyze* for EMERLD (see Section 3.3.5).

The following steps are required to build it:

1. In the folder with the source code run:
   `./gradlew build`
2. The built application can be found as archives at:
   `codyze-cli/build/distribution/`

3.3.3 Instructions for use

Instructions for use are provided as part of the released *Codyze* for EMERALD package (cf. folder '`docs/`' in Table 3) and are included in *Codyze*'s public GitLab repository[18].

[17] https://adoptium.net/de/temurin/releases/?version=17&package=jdk
[18] https://git.code.tecnalia.com/emerald/public/components/codyze

Figure 3 illustrates how evidence extractors will be used for setting up certification targets in the EMERALD UI. Please refer to D4.3 [14] for further visualizations of the EMERALD UI.

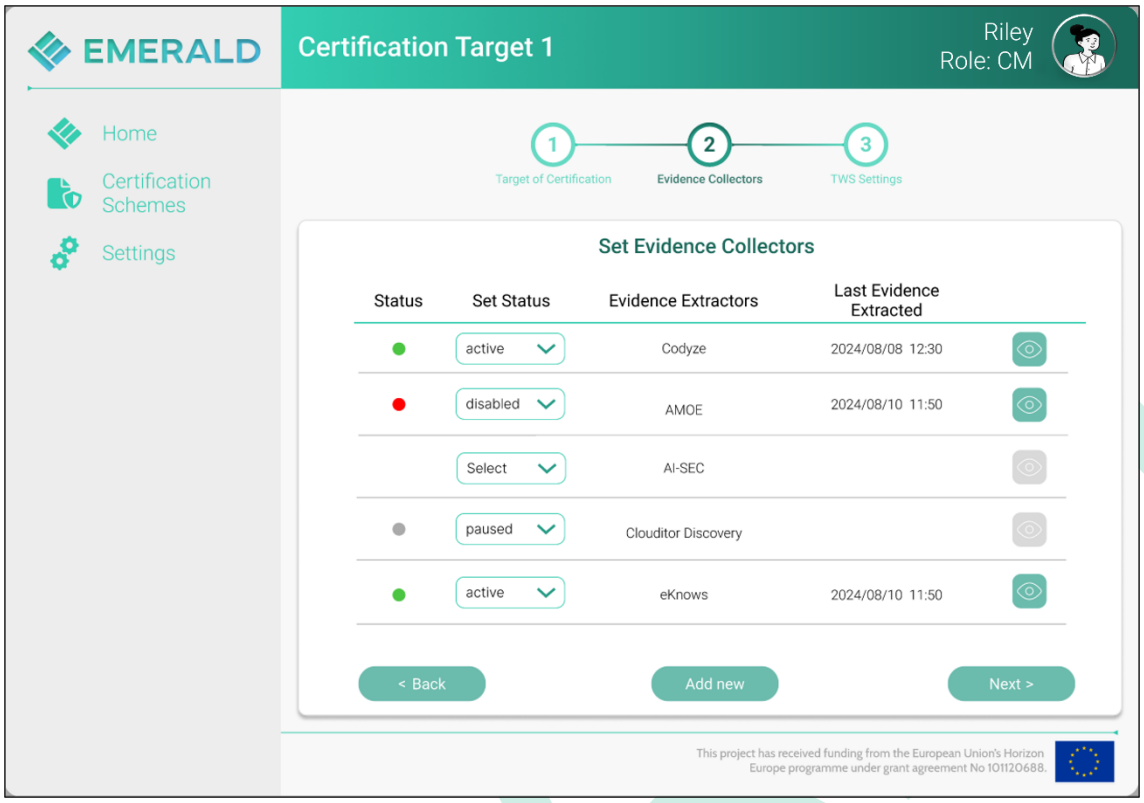

Figure 3. Configuration of evidence collectors in the EMERALD UI (D4.3 [14])

3.3.4 Licensing information

Codyze for EMERALD and its subcomponents are licensed as open source under Apache License, Version 2.0. In addition, it is ensured that third-party dependencies are compatible with the Apache License, Version 2.0. In particular, the main *Codyze* dependency for source code analysis, the CPG, is also licensed as open source under Apache License, Version 2.0.

3.3.5 Download

Codyze for EMERALD is available from the public EMERALD GitLab repository[18] hosted by TECNALIA. The repository will host the source code, the documentation and the binary artefacts consisting of a container image and a release archive.

3.4 Limitations and future work

Codyze for EMERALD is based on *Codyze* for MEDINA and aims to reach TRL 6-7. Changes in the EMERALD framework compared to the MEDINA framework require adjustments to *Codyze*. Primarily, *Codyze* for MEDINA reported evidence and assessment results. In contrast, the EMERALD framework uses a common taxonomy of evidence to create a Certification Graph (*CertGraph*) to coalesce all information from gathered evidence. As a result, *Codyze* for EMERALD can report only evidence classified according to the taxonomy and ontology of the *CertGraph*. The assessment of the information is dedicated to the assessment component within EMERALD. These changes require a change in the evidence collection and classification within *Codyze* and are part of the ongoing work in EMERALD WP2, Task 2.2.

4 *eknows evidence extractor*

eknows[19] [15] [16] is a software analysis platform developed by SCCH. The platform supports rapid development of multi-language software analysis tools from pre-built modules, which build on a technology-agnostic, generic layer and are extended to meet use case-specific requirements.

4.1 Functional description

Overall purpose. To efficiently create knowledge extraction techniques delivering required evidence to verify if application source code complies to security requirements, we rely on the multi-language software platform *eknows.* This platform quickly and flexibly supports the creation of evidence extraction functions by reusing prefabricated parsing, analysis, and generation modules. *eknows* provides support for main reverse engineering activities, i.e., knowledge extraction, transformation, analysis, and generation (visualization) as depicted in Figure 4). The cornerstone of the platform implementation is a generic programming language-independent representation of source code that can be reused across analysis and generation modules to prepare suitable evidence.

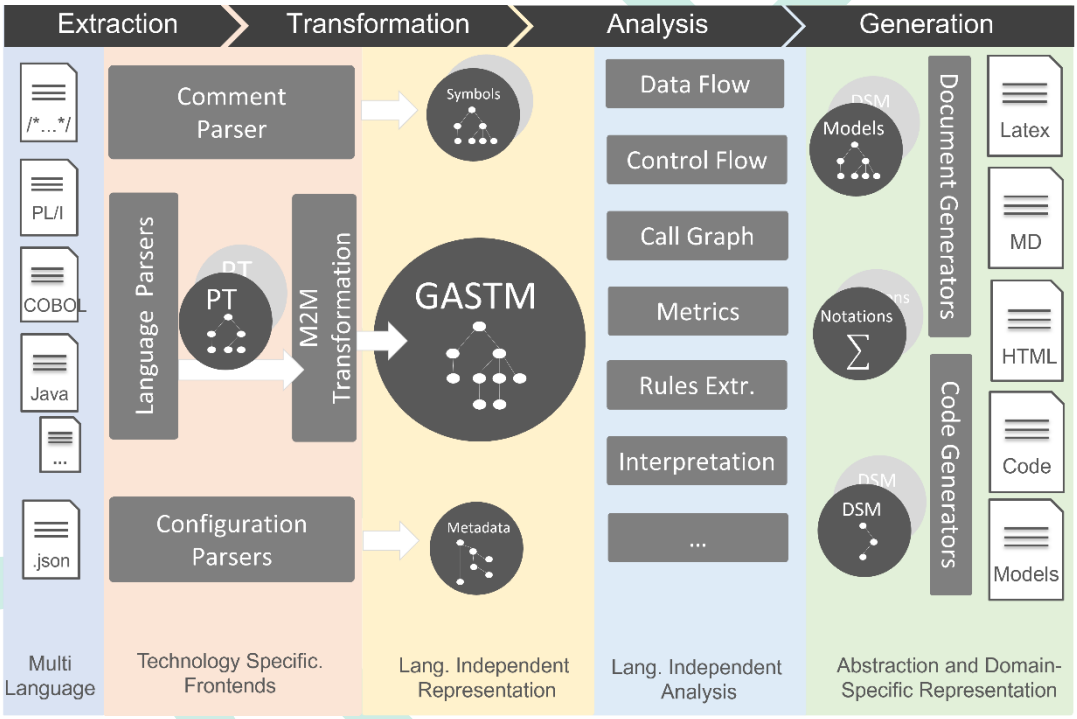

Figure 4. Reverse engineering activities supported by the software platform eknows [15]. Further explanations of subcomponents are provided in Section 4.2.1.1.

Context and scope. While the development of the platform was driven by domain-specific requirements from various stakeholders, e.g., business analysts or software architects, and multi-technology use cases, an architecture that supports reuse of modules for the analysis of software and generation of artefacts from different programming languages was envisaged from the beginning.

The delivered prototype builds upon selected modules of the platform (*eknows core*) and provides a "wrapper" containing the newly developed functions for EMERALD to extract security-related evidence from application source code (*eknows evidence extractor).* Thereby,

[19] https://www.scch.at/software-science/projekte/detail/eknows

generic modules for the model-guided symbolic execution of use case-specific conformity checks and fact extraction are extended. Also, generic modules for business rule extraction for domain-specific rule and constraint localization will be extended. New analyses will be necessary to break down high-level security controls from catalogues, such as EUCS or BSI C5, into checkable source code properties. New generation functions to create evidence based on these source code properties and to integrate them into *CertGraph* will be provided. Which concrete modules are needed from *eknows core,* and which new parts will be added is not yet clear at the time of writing. It depends on which security controls are derived from the security catalogues, and which programming languages are used in the cloud application code. Details on this scope will be described in the next deliverable D2.3 [5].

Motivation. As already said, *Codyze* and *eknows* will not be technically integrated, as both can co-live in the EMERALD CaaS framework. The motivation to include *eknows* in the EMERALD framework is, on the one hand, to demonstrate that the framework is open to extension and not tight to a specific tool. On the other hand, the coverage of security controls should be increased by including *eknows* as an additional source extractor in the EMERALD framework to be able to check more comprehensively whether the available source code conforms to the selected security controls.

Requirements. The relevant requirements from D1.3 [12] with their respective implementation state (partially / fully /not implemented) and a brief description of how they are / will be implemented are given in tables from Table 4 to Table 8.

Table 4. EKNOWS.01 - Integration into existing systems

Field	Description
Requirement ID	EKNOWS.01
Short title	Integration into existing systems
Description	The component should be integrable into existing systems, development environments and workflows, for example by using APIs like REST by compatibility with CI/CD-Pipelines.
Status	Accepted
Priority	Must
Component	eknows
Source	Component
Type	Technical
Related KR	KR1_EXTRACT
Related KPI	KPI 1.1
Validation acceptance criteria	The availability of the API will be tested via an OpenAPI client.
Progress	Partially implemented – 30%
Milestone	MS3: Integrated audit suite V1 (M18)

The prototype currently offers a command line interface, which can be integrated in a flexible way.

Table 5. EKNOWS.02 - Resilience while analysing erroneous code

Field	Description
Requirement ID	EKNOWS.02
Short title	Resilience while analysing erroneous code
Description	The source code analysed by the component could be erroneous, for example syntactical and semantical errors could be encountered

	while parsing it. Furthermore, an unknown dialect of a language could be encountered. An appropriate error handling strategy for such situations is necessary: Erroneous code will be skipped and not be further analysed. A corresponding error message will be stored in the gathered evidence.
Status	Accepted
Priority	Should
Component	eknows
Source	Component
Type	Technical
Related KR	KR1_EXTRACT
Related KPI	KPI 1.1
Validation acceptance criteria	The component will receive erroneous source code. Processing should run through, and a corresponding error message should be found in the generated evidence.
Progress	Partially implemented – 70%
Milestone	MS5: Components V2 (M24)

Some error cases are shown to the user.

Table 6. EKNOWS.03 - Multi-language support

Field	Description
Requirement ID	EKNOWS.03
Short title	Multi-language support
Description	The component should be able to analyse source code written in different programming languages and should support at least Java and Python.
Status	Accepted
Priority	Must
Component	eknows
Source	Component
Type	Technical
Related KR	KR1_EXTRACT
Related KPI	KPI 1.1
Validation acceptance criteria	The component will receive source files written in Java and Python and should be able to process each language and generate an output.
Progress	Partially implemented – 50%
Milestone	MS5: Components V2 (M24)

Currently, the Java frontend is used.

Table 7. EKNOWS.04 - Support EMERALD evidence format

Field	Description
Requirement ID	EKNOWS.04
Short title	Support EMERALD evidence format
Description	The analysis of results is offered in a structured and standardized format, the EMERALD evidence format (see data model in [9]). This enables further processing and queries in other components.
Status	Accepted
Priority	Must

Component	eknows
Source	Component
Type	Technical
Related KR	KR1_EXTRACT
Related KPI	KPI 1.1
Validation acceptance criteria	The component receives source code to analyse and generates an output from it. This output will be validated against the schema of the evidence format.
Progress	Not implemented – 0%
Milestone	MS3: Integrated audit suite V1 (M18)

Simple JSON output is generated so far.

Table 8. EKNOWS.05 - Static code analysis

Field	Description
Requirement ID	EKNOWS.05
Short title	Static code analysis
Description	The component uses static code analysis methods. Such methods are, for example, data flow analysis, call graph analysis, symbolic execution, or control flow analysis. One or multiple methods (possibly in combination) will be used to gather evidence. The actual used method(s) depend(s) on the metric, for which evidence should be extracted.
Status	Accepted
Priority	Must
Component	eknows
Source	Component
Type	Technical
Related KR	KR1_EXTRACT
Related KPI	KPI:1.1
Validation acceptance criteria	Code review: Review code and check if static code analysis methods are used/implemented.
Progress	Partially implemented – 60%
Milestone	MS5: Components V2 (M24)

Currently, symbolic execution is used.

Innovation. In addition to the described innovation of providing compliance by design through source code analysis in cloud applications in Section 3.1, using a multi-language software platform for rapid development of evidence extraction techniques from pre-built analysis and generation modules for certification of cloud applications is a big advancement. Following the extract-abstract-view metaphor [17] that can be considered as reference architecture for generating suitable evidence for security metrics in EMERALD, as well as using a standard-based approach (i.e., the abstract syntax tree metamodel (ASTM)[20] of the Object Management Group (OMG)) as the generic representation of the parsed source code, is another notable aspect.

[20] https://www.omg.org/spec/ASTM/1.0/About-ASTM

4.2 Technical description

4.2.1 Prototype architecture

Figure 5 shows an overview (not fully complete) of the consisting modules of the *eknows* platform. At the bottom of the figure, various frontends using specific parsers for extracting knowledge from different programming languages, such as Java, C#, and PL/SQL, are depicted. The generic representation of the parsed source code based on the ASTM metamodel builds the basis for a set of analyses, such as *Call Graph*, *Control Flow*, and *Dependency*. Various generation modules for visualizing application code as reports or diagrams form the top of the platform.

As already mentioned, only a selected subset of these building blocks is reused / extended in the delivered prototype and additional functions will be developed. The prototype architecture will be improved and detailed as the concrete security controls and metrics are available, which will be reported in the next version of this deliverable, namely D2.3 [5].

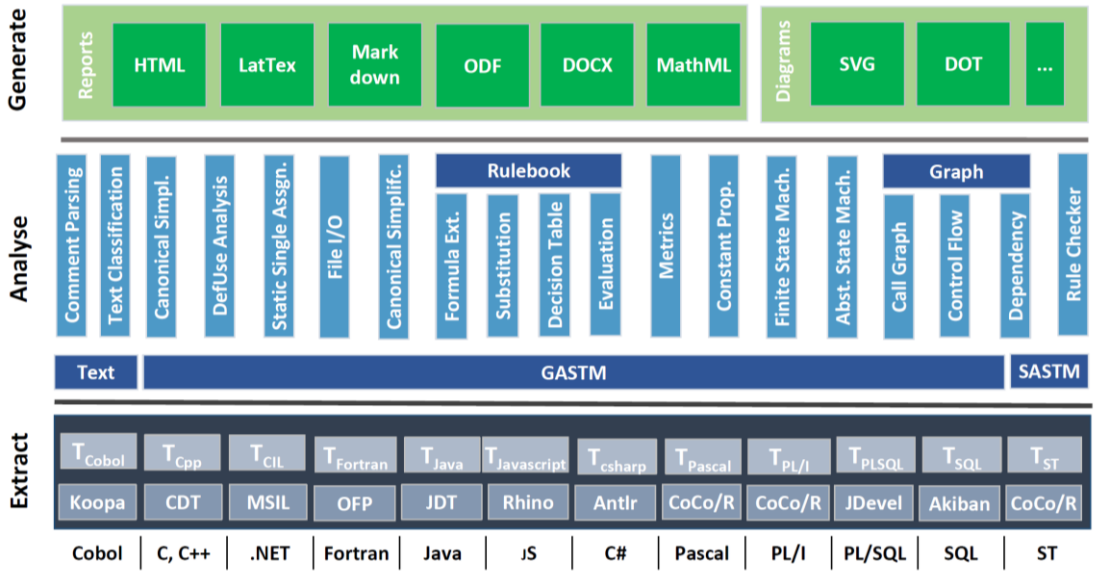

Figure 5. Overview of eknows platform building blocks [16]

4.2.1.1 Subcomponents description

The *eknows* platform provides pre-built modules that facilitate 1) language parsing and transformation of code into a generic abstract syntax tree (AST), 2) structural and behavioural analysis of software, and 3) reporting and visualization of analysis results (see Figure 5). Software solutions built on top of *eknows* integrate required modules as-is and add functionality required for specific use cases.

Extraction modules, also referred to as language frontends, parse information from software systems (i.e., source code and comments) and transform parse trees into ASTs and generic data structures used in analysis. *eknows* currently supports over 14 programming languages. To provide robust and up-to-date parsing infrastructure, *eknows* reuses freely available parsing components as far as possible. For instance, to parse Java or C++, Eclipse JDT and Eclipse CDT are used. If no ready-to-use source code parser is available, parser generators (i.e., ANTLR and CoCo/R) are used to generate parsing infrastructure from context-free grammar specifications. In the delivered prototype we will most likely rely on the Java and Python frontends.

To reuse analysis components across different technology stacks, *eknows* builds upon the ASTM. The standard provided by the OMG is used as common intermediate representation and is

composed of the generic AST metamodel (GASTM) and a set of complementary, language-specific specifications, called the specialized AST metamodels (SASTM). To reuse analysis components as far as possible, language-dependent models are kept to a minimum. In the delivered prototype we will reuse and extend existing analysis, such as the *symbolic evaluation*, and will develop new analysis depending on the selected security controls and metrics, e.g., the *TLS analysis*.

Finally, *eknows* provides a set of visualization and reporting components. Generated elements fall into the categories text, mathematical formula, tables, charts, or graphs. Input for these modules are ASTM data structures or specific result data structures created by analysis modules. Graphs are output in intermediate format, e.g., Graphviz DOT. To generate documentation, a format-independent data structure is used to specify document structure and content (e.g., sections, paragraphs, formulas, figures, or tables). Document specifications can be output as LaTeX, HTML, Markdown, and Open Document Format (ODF). For reporting evidence in the delivered prototype, we will refine the existing functionality to generate the results in the defined EMERALD format.

4.2.2 Technical specifications

The Java-based platform (*eknows* core) comprises over 350K source lines of code (SLOC). It uses libraries as dependencies to implement its functionality. For source code parsing, *eknows* uses language-specific parsers (see Table 9) traditionally built with compiler-generator tools. A central point is the generic representation of the parsed content. The dependencies are managed using Maven[21].

Table 9. Supported programming languages by eknows

Language	Parser	Supported version
Adele	CoCo/R[22]	n.a.
B&R (Bernecker + Rainer)	CoCo/R	n.a.
C, C++	Eclipse CDT[23]	C++17
C#	ANTLR[24]	C# 7.0
CIL/.NET	n.a.	.NET 4.5
COBOL	Koopa[25]	COBOL 85
Codesys/Bachmann	Xtext[26]	3
Fortran	Open Fortran[27]	77/2003
JCL	CoCo/R	JCL z/OS 2.2
Java	Eclipse JDT[28]	Java 8
Javascript	Mozilla Rhino[29]	ES 6
Natural	CoCo/R	v 4.2.6
Oberon	CoCo/R	n.a.
Pascal	CoCo/R	Pascal 7.0
PL/I	CoCo/R	z/OS 4.1

[21] https://maven.apache.org/
[22] https://ssw.jku.at/Research/Projects/Coco
[23] https://projects.eclipse.org/projects/tools.cdt
[24] https://www.antlr.org/
[25] https://github.com/krisds/koopa
[26] https://eclipse.dev/Xtext/
[27] https://github.com/OpenFortranProject/open-fortran-parser
[28] https://www.eclipse.org/jdt
[29] https://github.com/mozilla/rhino

Language	Parser	Supported version
PL/SQL	JDeveloper/Akiban[30]	9.1
Python	ANTLR	Python 3.6
Sigmatek	ANTLR	n.a.
SQL	Akiban	MySQL 5.6

The delivered prototype, the *eknows evidence extractor*, is based on *eknows core* and reuses modules as far as possible.

The *eknows evidence extractor* consists of an executable binary distribution and runs stand-alone. Like *Codyze*, the *eknows evidence extractor* is executed on source code of cloud applications and services. The integration of the *eknows evidence extractor* at the CSPs requires a platform for CI/CD. The *eknows evidence extractor* can be integrated into CI/CD pipelines by using the binary distribution. In the current state it is not yet integrated with other components. Currently, a CLI is provided, and a REST interface can be provided in the future, if needed. Findings are generated as console output. This output will be submitted to the *Evidence Store* of the EMERALD framework in the specified data format following the terms defined in the *CertGraph* ontology.

4.3 Delivery and usage

The following subsections detail the delivery and usage of *eknows* for EMERALD. The provided information is currently work in progress and may change in the future.

4.3.1 Package information

The *eknows evidence extractor* is developed as a Java application with the support of Maven[31] as build tool. Table 10 shows the structure of the Gitlab repository[32] and its contents.

Table 10. Overview and description of package structure for the eknows evidence extractor

Folder	Description
`eknows/`	The prebuilt *eknows* binaries should be placed here. In addition, installation scripts are provided in this folder.
`src/`	Source code root.
`src/main/`	Source code for the *eknows evidence extractor*.
`src/test/`	Source code for unit tests of the *eknows evidence extractor*.
`testfiles/`	Test files, which are used for unit tests and can be used for demo purposes as well.
`LICENSE`	License text (Apache License, Version 2.0).
`README.md`	Compact guide on how to build and use the extractor.

4.3.2 Installation

Requirements:

- Java 17 JDK[33].
- Maven[31].
- Source code of the *eknows evidence extractor* (see Section 4.3.5).

[30] https://github.com/brunoribeiro/sql-parser
[31] https://maven.apache.org/download.cgi
[32] https://git.code.tecnalia.com/emerald/public/components/eknows
[33] https://adoptium.net/de/temurin/releases/?version=17&package=jdk

The following steps are required to build the *eknows evidence extractor*:

1. Install *eknows core* for EMERALD (this step is only required when building for the first time or when *eknows core* for EMERALD is updated).
    a. Get the *eknows-core-for-emerald* JAR file from the internal repository (see Section 4.3.5).
    b. Add the JAR file to the *eknows* folder.
    c. Run *install.cmd* or *./install* within the *eknows* folder.

2. Build-
    a. Run *mvn package -DskipTests.*

3. The built extractor can be found at *target/eknows-evidence-extractor-<version>-jar-with-dependencies.jar.*

4.3.3 Instructions for use

The *eknows evidence extractor* can be run from the command line for now by invoking:

java -jar target/eknows-evidence-extractor-<version>-jar-with-dependencies.jar -f <file to analyse>

This will generate a JSON output, which contains the analysed file name and, in the current development status, the detected TLS versions.

Figure 3 illustrates how evidence extractors will be used for setting up certification targets in the EMERALD UI. Please refer to D4.3 [14] for further visualizations of the EMERALD UI.

4.3.4 Licensing information

The licensing is split into two parts:

1. The *eknows evidence extractor*, which is developed in the context of EMERALD, is licensed under Apache 2.0[34] and will be made available to the public as open-source software.
2. The foundation of the extractor, *eknows core*, is closed source and binaries are made available to the EMERALD project consortium within the context of the project under the *eknows Binary Usage Software License* (see *Appendix A: eknows Binary Usage Software License*).

4.3.5 Download

The *eknows evidence extractor* is available from the public EMERALD GitLab repository[35] hosted by Tecnalia. The repository is going to host the source code and the documentation.

The binaries for *eknows core* are available to the EMERALD project consortium in a separate private EMERALD GitLab repository[36].

4.4 Limitations and future work

Currently, the extractor just supports the analysis of source code written in Java and at least one additional programming language will be added in the future, however the *eknows* platform

[34] http://www.apache.org/licenses/LICENSE-2.0
[35] https://git.code.tecnalia.com/emerald/public/components/eknows
[36] https://git.code.tecnalia.com/emerald/private/components/eknows/eknows-core-for-emerald [internal use only - authentication required]

already supports a variety of programming languages, which can be integrated to the extractor if needed.

Further, the *eknows evidence extractor* currently provides one use case (the extraction of the configured TLS versions). Depending on the certification requirements, a more extensive analysis must be implemented.

Finally, the *eknows evidence extractor* will be fully integrated and deployed in the EMERALD environment.

5 Conclusions

The EMERALD project proposes a holistic approach to evidence collection, focusing on all levels of the cloud service, including the infrastructure layer (e.g., virtual resources) to the business layer (e.g., policies and procedure), and the implementation layer (e.g., source code files).

In this deliverable, which is the initial output of Task 2.2, we presented the technical report about the design, architecture, and current implementation states of EMERALD source evidence extraction components. The components follow the overall EMERALD framework approach and are aligned with the technical requirements gathered in the scope of WP1. This report presents the relation of the presented components with the other parts of the EMERALD framework and details the individual components' internal structure, their subcomponents, and information about their technical implementation.

The components presented in this document include two evidence extractors supporting the assessment of the security and compliance of a cloud application's source code, i.e., *Codyze* and *eknows evidence extractor*. At this point of the project, the components, based on some background works, have already working prototypes that can be (partially) integrated with some other EMERALD components and satisfy some of their respective requirements, as expressed in D3.1 [8]. Future work will also deal with complying to the needs of business requirements of the selected security controls developed in WP5.

In the further course of the project, the source evidence extractors *Codyze* and *eknows evidence extractor* will be integrated into the EMERALD framework. The subsequent and final iteration of this report (D2.3 [5]) will provide the progress of the updated components in project month 24.

6 References

[1] EMERALD Consortium, "D2.4 AMOE – v1: Evidence extraction from policy documents that can be integrated with the certification graph," 2024.

[2] EMERALD Consortium, "D2.6 ML model certification – v1: Security and privacy preserving evidence that can be integrated with the certification graph," 2024.

[3] EMERALD Consortium, "D2.8 Runtime evidence extractor – v1: Evidence extraction from runtime data that can be integrated with the certification graph," 2024.

[4] EMERALD Consortium, "D2.1 Graph Ontology for Evidence Storage: Description of a uniform schema for storing and linking heterogenous data," 2024.

[5] EMERALD Consortium, "D2.3 Source Evidence Extractor – v2: Evidence extraction from source code that can be integrated with the certification graph," 2025.

[6] EMERALD Consortium, "D2.10 Certification Graph– v1: Integration of the graph with semantically linked and combined evidence," 2025.

[7] EMERALD Consortium, "D2.11 Certification Graph– v2: Integration of the graph with semantically linked and combined evidence," 2026.

[8] EMERALD Consortium, "D3.1 Evidence Assessment and Certification - Concepts - v1," 2024.

[9] EMERALD Consortium, "D1.1 Data modelling and interaction mechanisms - v1," 2024.

[10] EMERALD Consortium, "EMERALD Glossary in D1.3- EMERALD solution architecture - v1," 2024.

[11] EMERALD Consortium, "D3.3 Evidence Assessment and Certification-Implementation-v1," 2024.

[12] EMERALD Consortium, "D1.3 EMERALD solution architecture - v1," 2024.

[13] K. Weiss and C. Banse, "A Language-Independent Analysis Platform for Source Code," arXiv, 2022.

[14] EMERALD Consortium, "D4.3 User interaction and user experience concept – v1," 2024.

[15] V. Geist, M. Moser, J. Pichler and F. Schnitzhofer, "Innovating Industry with Research: eknows and Sysparency," *IEEE Software,* pp. 1-7, 2024.

[16] M. Moser and J. Pichler, "eknows: Platform for multi-language reverse engineering and documentation generation," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 559-568, 2021.

[17] C. Lange, H. Sneed and A. Winter, “Comparing graph-based program comprehension tools to relational database-based tools,” in *9th International Workshop on Program Comprehension (IWPC 2001)*, pp. 209-218, 2001.

Appendix A: eknows Binary Usage Software License

This Software Usage License ("License") is granted to all partners of the EMERALD project consortium ("Licensee") by Software Competence Center Hagenberg GmbH, a legal entity organized and existing under the laws of Austria, having its principal place of business at Softwarepark 32a,4232 Hagenberg, Austria ("Licensor").

1. Scope of Use:

The Licensee is granted a non-exclusive, non-transferable license to use java binaries (jar) of the eknows software platform (hereinafter referred to as "the Software") solely for development, piloting, and evaluation purposes within the EMERALD project context. Usage of the Software for any other purpose or in any other context is strictly prohibited.

2. Restrictions:

a. Licensee shall not modify, distribute, sublicense, or transfer the Software to any third party.

b. Licensee shall not use the Software outside the project context defined above.

c. Licensee shall not reverse engineer, decompile, or disassemble the Software.

3. Intellectual Property:

All intellectual property rights, including but not limited to copyrights, patents, and trade secrets, in and to the Software remain the sole property of the Licensor.

4. Term and Termination:

This Agreement shall be effective from the date of acceptance and shall continue until terminated by either party. Licensor may terminate this Agreement immediately upon breach of any term herein.

5. Governing Law:

This Agreement shall be governed by and construed in accordance with the laws of Austria.

6. Limitation of Liability:

In no event shall the Licensor be liable for any special, indirect, incidental, consequential, or exemplary damages, including, but not limited to, loss of profits or data, arising out of or in connection with the use or performance of the Software, even if Licensor has been advised of the possibility of such damages.

Licensor: Software Competence Center Hagenberg GmbH Softwarepark 32a 4232 Hagenberg Austria
Tel.: +43 50 343 E-Mail: office@scch.at Web: http://www.scch.at

scch 2024