EMERALD

Deliverable D2.3

Source Evidence Extractor – v2

Editor(s):	Verena Geist
Responsible Partner:	Software Competence Center Hagenberg GmbH (SCCH)
Status-Version:	Final-v1.1
Date:	15.09.2026
Type:	OTHER
Distribution level:	PU

Project Number:	101120688
Project Title:	EMERALD

Title of Deliverable:	D2.3 – Source Evidence Extractor – v2
Due Date of Delivery to the EC	31.10.2025
Updated Version of Deliverable	15.09.2026

Workpackage responsible for the Deliverable:	WP2 – Methodology for knowledge extraction
Editor(s):	Verena Geist (SCCH)
Contributor(s):	Florian Wendland (FHG) David Pizhuk (SCCH) Stefan Schöberl (SCCH)
Reviewer(s):	Martí Fabregat Pous (CXBTECH) 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 the final version on source evidence extractors. Changes to the first/interim version D2.2 are highlighted.
Keyword List:	Knowledge extraction, source code files, technical evidence, static code analysis, *Codyze*, *eknows-e3*.
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	16.09.2025	First draft version, ToC	Verena Geist (SCCH)
v0.2	30.09.2025	Update of eknows-e3 requirement status	Stefan Schöberl (SCCH)
v0.3	01.10.2025	Highlight of changes to previous document version	Verena Geist (SCCH)
v0.4	06.10.2025	New developments on eknows-e3 (functional and technical descriptions)	David Pizhuk (SCCH)
v0.5	08.10.2025	Update of Codyze, highlight of changes and advancements	Florian Wendland (FhG)
v0.6	08.10.2025	*eknows-e3* architecture diagram	David Pizhuk (SCCH)
v0.7	08.10.2025	Update of terms and abbreviations, executive summary, introduction, conclusion, and references.	Verena Geist (SCCH)
v0.8	21.10.2025	Internal review	Martí Fabregat Pous (CXB)
v0.9	24.10.2025	Revision of internal review	Verena Geist (SCCH)
v0.10	29.10.2025	Revision of internal review	Florian Wendland (FhG)
v0.11	29.10.2025	Final reviewed version	Juncal Alonso, Cristina Martinez (TECNALIA)
v0.12	30.10.2025	Revision of internal review	Verena Geist (SCCH) Florian Wendland (FhG)
v1.0	31.10.2025	Submitted to the European Commission	Juncal Alonso, Cristina Martínez (TECNALIA)
v1.0.1	24.04.2026	Relevant updates in development of *eknows-e3*	David Pizhuk, Stefan Schöberl (SCCH)
v1.0.2	15.06.2026	Relevant updates in development of *Codyze*	Florian Wendland (FhG)
v1.0.3	23.06.2026	Refinement and review of updates	Verena Geist (SCCH)
v1.0.4	23.06.2026	Internal review of updated version	Martí Fabregat Pous (CXBTECH)
v1.0.5	29.07.2026	Revision of internal review of updated version	Stefan Schöberl (SCCH) Florian Wendland (FhG)
v1.0.6	04.08.2026	Final reviewed version of updated version	Juncal Alonso, Cristina Martinez (TECNALIA)
v1.1	15.09.2026	Updated version submitted to the European Commission	Juncal Alonso, Cristina Martínez (TECNALIA)

Table of contents

List of tables

List of figures

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
BCJSSE	BouncyCastleJSSE
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
CSP	Cloud Service Provider
DOT	Markup-Language
DSL	Domain-Specific Language
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-e3	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
GAST	Generic AST
GASTM	Generic AST Metamodel
HTML	Hypertext Markup Language
HTTPS	Hypertext Transfer Protocol Secure
ID	Identifier
JAR	Java Archive
JCL	Job Control Language
JDT	Java Development Tools
JNA	Java Native Access
JSON	JavaScript Object Notation
JSSE	Java Secure Socket Extension
Koopa	(COBOL) Parser Generator
KPI	Key Performance Indicator
KR	Key Result
LLMs	Large Language Models
LTS	Long-Term-Support
MAC	Message Authentication Code
MD	Markdown

MEDINA	Predecessor project of EMERALD
OAuth	Open Authorization
ODF	Open Document Format
OMG	Object Management Group
OSV	Open Source Vulnerability
PL/I	Programming Language One
PL/SQL	Procedural Language/Structured Query Language
POM	Project Object Model
PT	Parse Tree
REST	Representational State Transfer
SARIF	Static Analysis Results Interchange Format
SASTM	Specialized AST Metamodel
SE	Standard Edition
SSH	Secure Shell
SSL	Secure Sockets Layer
SVG	Scalable Vector Graphics
SW	Software
TLS	Transport Layer Security
TRL	Technology Readiness Level
URL	Uniform Resource Locator
WP	Work Package

Executive Summary

This deliverable presents the final design, architecture, and implementation state of the source evidence extractors of WP2, i.e., *Codyze* and *eknows-e3*. 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. Evidence is prepared according to the integrated, graph-based model of semantically linked and combined evidence, provided in D2.11 [1]. The extracted evidence is stored and assessed, i.e., to verify the implementation of security metrics, in the scope of WP3.

EMERALD follows a knowledge graph-based approach to provide a unified view of the cloud service under evaluation 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 24 (October 2025), provide functional and technical details on further evidence extractors from different sources, i.e., D2.5 [2] on evidence extraction from policy documents in Task 2.3, D2.7 [3] on security and privacy preserving evidence extraction in Task 2.4, and D2.9 [4] on runtime data extraction in Task 2.5.

This document starts by highlighting updates from D2.2 [5], the initial version of this deliverable, and advancement to the predecessor project MEDINA. It then illustrates how the source evidence extractors are integrated into the overall EMERALD architecture. The main part provides functional and technical descriptions of the two extractor components, *Codyze* and *eknows-e3*, including their purpose and scope, the 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 and conclusions.

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 evaluation 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*) [1], 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 final design and implementation of the EMERALD evidence extractors developed in *Task 2.2 Extraction of evidence / knowledge out of source code.* Evidence on the source code level is primarily gathered by the source evidence extractors *Codyze* and *eknows-e3*, 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-e3*. *eknows* offers language-independent analyses and can be extended for various use cases. A compact overview of both source code extractors in the form of *Components Cards* can be found in D1.4 [6].

Note that the integration of *Codyze* and *eknows-e3* has not been intended. The EMERALD framework works with more than one source evidence extractor, i.e., a security metric may work well with *Codyze*, and another using *eknows-e3*. Importantly, the resulting evidence format is 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* [1].

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 *eknows-e3*. For each source extractor, functional and technical descriptions are provided, including their purpose and scope, the coverage of the EMERALD requirements, the components' internal architecture, their subcomponents, and details about the programming language, libraries, etc.

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

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 also 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 *eknows-e3*.

1.3 Updates from D2.2

This deliverable evolves from D2.2 [5], and with the ultimate goal of making the document self-contained and easier to follow, part of the content comes from D2.2 [5] since it has not changed, and other parts are new. To simplify tracking progress and updates from the previous versions, *Table 1* shows a summary of the main changes and additions to each section of the document. Please also note that the latest version v1.1 of this deliverable reflects the final status of development of the source evidence extractors until the end of WP2.

Table 1. Overview of deliverable updates with respect to D2.2 [5]

Section	Changes
Overall document	• The source evidence extractor *eknows* was renamed to *eknows-e3* • Updated terms and abbreviations • Updated references
1 Introduction	• Added new subsection *1.3 Updates from D2.2* • Added new subsection *1.4 Technological Advances from the MEDINA project*
2 Source evidence extractors in the EMERALD architecture	• Updated component overview diagram • Updated current versions of deliverables
3 *Codyze* for EMERALD	• Updated requirement status • Updated architecture diagram • Updated technical specifications • Included connection to the *Trustworthiness System* component • Updated limitations and future work
4 *eknows evidence extractor* (*eknows-e3*)	• Updated functional description to reflect the current development status of *eknows-e3*, including integration of different analysis modules, a resolving mechanism, and generation functionalities to integrate evidence into *CertGraph* • Updated requirement status • Updated prototype architecture *for eknows-e3* • Updated internal structure of *eknows-e3* • Added three major subcomponents of *eknows-e3* (i.e., for extracting and interpreting evidence, sending evidence, and integration into the CI/CD environment)

Section	Changes

	• Added overview figure of the restructured *eknows-e3* modules and their dependencies
	• Updated and extended technical specifications:
	  ○ Added Gradle integration
	  ○ Added vulnerability analysis
	  ○ Added extraction of Git-related security evidence
	  ○ Included SSL/TLS libraries (with extraction of protocols and cipher suites)
	• Updated and extended package information
	• Updated installation instructions and instructions for use
	• Added required environment variables for configuration
	• Added supported CLI arguments for configuration
	• Added versioning support
	• Updated limitations and future work on the integration of static code analysis tools into coding agents, and open questions and challenges for *eknows-e3*

5 Conclusions	• Updated conclusions on source evidence extractors

1.4 Technological Advances from the MEDINA project

EMERALD builds upon the outcomes of the MEDINA project: *Security framework to achieve a continuous audit-based certification in compliance with the EU-wide cloud security certification scheme* (GA 952633). While MEDINA only reached TRL 5, the goal of EMERALD is to enhance the system and elevate its TRL from prototype (TRL5) to product (TRL7). This will be achieved by:

- making AI and high-level technologies approachable for non-experts,
- increasing the abstraction level in evidence extraction across various layers of the cloud service (infrastructure, code, business processes) in an easy-to-use and understandable way, and
- providing trustworthy support for continuous auditing processes.

To facilitate this, we develop an interaction concept that connects various components through a new UI/UX for the overall EMERALD product and ensures alignment with the auditors' workflow.

In MEDINA, ***Codyze*** implemented its own assessments, leading to increased maintenance efforts when requirements changed over time. Therefore, centralizing assessment is one of the goals in EMERALD. This is achieved by delivering exclusively (or as far as possible) raw evidence to the *Evidence Store* [7]. Moreover, to facilitate an assessment of the raw evidence, evidence is labelled based on a common ontology giving rise to the EMERALD *CertGraph* [1].

In the following, all changes in the *Codyze* component from MEDINA to EMERALD are listed:

- Collect evidence from source code rather than evaluate assessments,
- send evidence to the *Evidence Store* rather than assessments to the *Assessment* component,
- integration of the common ontology for *CertGraph, and*
- update architecture to support evidence extraction.

eknows-e3 is a new component developed within the EMERALD project. Building on the *eknows* software analysis platform by SCCH and outcomes of the MEDINA project, *eknows-e3* introduces new methods to automatically extract, interpret, and deliver security-relevant evidence from source code. Integrated into EMERALD's workflow and UI/UX, it complements *Codyze* to support continuous, audit-based cloud certification by making software analysis and evidence generation more automated, accessible, and aligned with auditors' needs.

2 Source evidence extractors in the EMERALD architecture

This section describes how the source evidence extractors interact, at a conceptual level, with certain EMERALD components. Figure 1 shows the EMERALD high-level architecture as a component diagram [8]. 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" [6]. In contrast, a tool is a "software element that has several disparate functions and therefore can be composed by several components" [6]. Therefore, *Codyze* and *eknows-e3* 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., *Codyze*, *eknows-e3*, *AI-SEC, Clouditor-Discovery*, and *AMOE*, are represented at the bottom part of Figure 1. The source evidence extractor components *Codyze* (see Section 3) and *eknows-e3* (see Section 4), which obtain technical evidence from the analysis of the source code of cloud applications, are highlighted using a thick frame. *AMOE* [2], 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* [4], also originated from MEDINA, collects evidence about the secure configuration of cloud resources, with a focus on runtime data extraction in EMERALD. *AI-SEC* [3] is a newly developed component in EMERALD and analyses AI models for several key evidence regarding robustness against adversarial attacks, explainability, and fairness.

As already mentioned in Section 1.4, 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* [7]. 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* [1].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* [7].

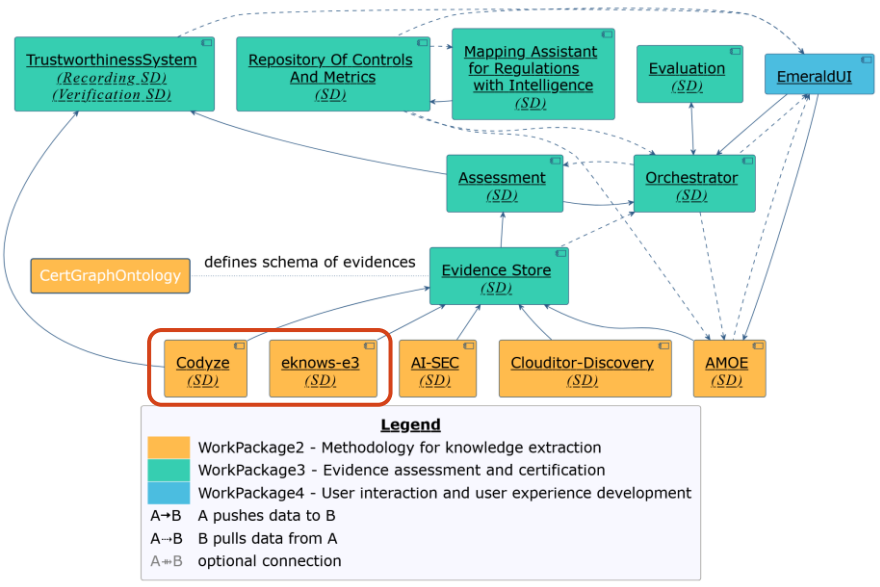

Figure 1. EMERALD component overview diagram [8]. The red rectangle highlights the source evidence extraction components, which are described in this deliverable

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. In addition, *Codyze* is supplemented by *Codyze-Provenance*, a runtime evidence extractor, that provides provenance between the source code analysed by *Codyze* and the artefacts generated in a CI/CD pipeline. *Codyze-Provenance* is described in D2.9 [4].

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. In addition, hashes of the pieces of evidence are also submitted to the *Trustworthiness System* [7] for cross-validation. *Codyze* uses the Blockchain client for the *TWS* and calls its API to post hashed information of evidence to the blockchain (cf. D3.4 [7]).

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 applications. 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 applications.

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 D1.4 [6] with their respective implementation progress (degree of implementation - *partially / fully / not implemented*), and a brief description of how they have been implemented are given in Table 2 and Table 3.

Innovation. *Codyze* provides 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.

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

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*.

Table 2. 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	Validated
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	100% - Fully implemented
Milestone	MS6: Integrated audit suite V2 (M30)
Note	Work depends in parts on and influences KR2_CERTGRAPH

Table 3. CODYZE.02 - Integration into CI/CD

Field	Description
Requirement ID	CODYZE.02
Short title	Integration into CI/CD pipelines for source code analysis
Description	*Codyze* should be made available as an easy to integrate step/task/component for a CI/CD pipeline. That way, *Codyze* is executed on code changes automatically as part of a CI/CD process.
Status	Validated
Priority	Should
Component	*Codyze*
Source	Component, KPI
Type	Technical
Related KR	KR1_EXTRACT
Related KPI	KPI 1.1
Validation acceptance criteria	*Codyze* can be integrated into CI/CD pipelines and can be executed on code changes
Progress	100% - Fully implemented
Milestone	MS6: Integrated audit suite V2 (M30)

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.

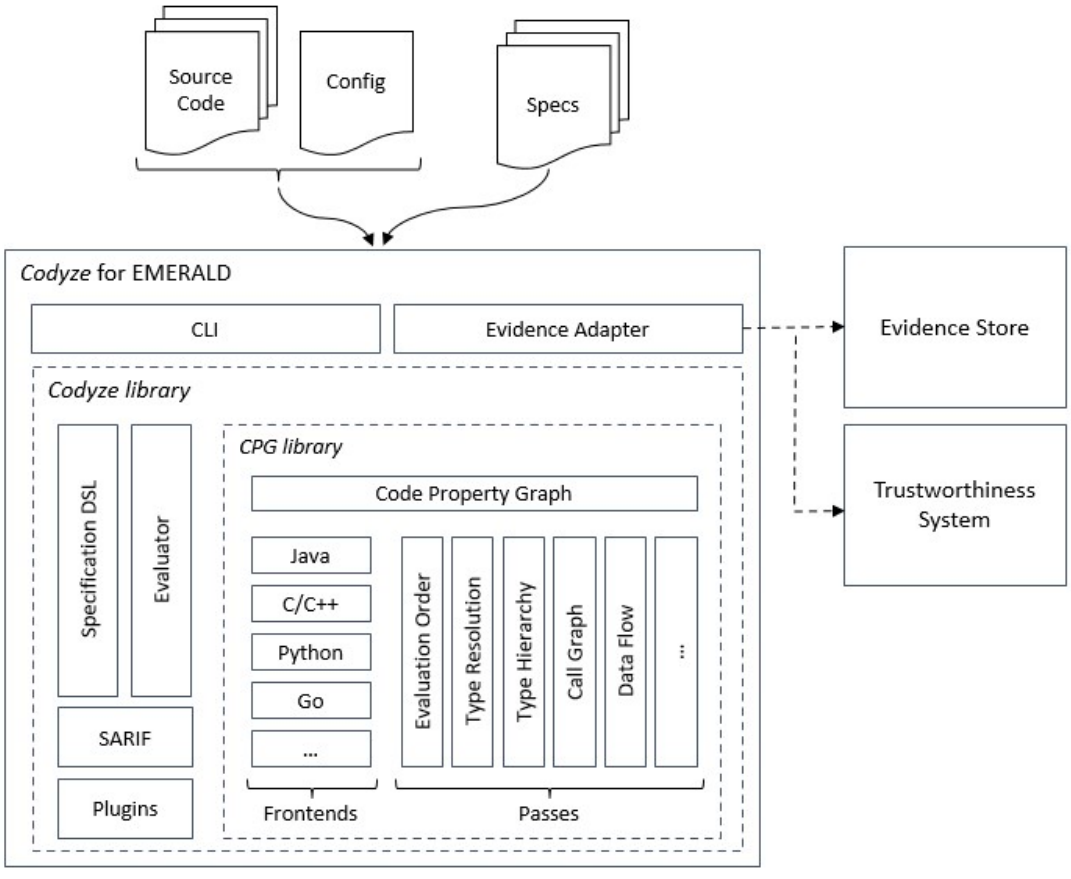

Figure 2. Architecture of Codyze for EMERALD, highlighting its modules and contributions within the EMERALD project (i.e., modules with dashed boxes are external)

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* and the *Trustworthiness System*. 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* used by *Codyze* for EMERALD is developed by Fraunhofer AISEC as an open-source project under the Apache License, Version 2.0, on GitHub[4]. 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[5] – a standard format to describe exchangeable results from static 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*[6] used by *Codyze* is developed by Fraunhofer AISEC as an open-source project under the Apache License, Version 2.0, on GitHub[7]. It is an implementation of a code property graph (CPG) [9]. 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.

Specifications. *Codyze* defines a domain-specific language (DSL) to allow the specification of compliance requirements as they relate to software implementations. As part of EMERALD,

[4] https://github.com/Fraunhofer-AISEC/cpg
[5] https://docs.oasis-open.org/sarif/sarif/v2.1.0/sarif-v2.1.0.html
[6] https://fraunhofer-aisec.github.io/cpg/
[7] https://github.com/Fraunhofer-AISEC/cpg

Codyze provides 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[8] using a Java Virtual Machine as execution platform. The build system uses Gradle[9] 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 4.

Table 4. Supported programming languages by Codyze through the CPG library

Language	CPG module	Parser	Supported version
Java	`cpg-language-java`	JavaParser[10]	Java SE 21
C++	`cpg-language-cxx`	Eclipse CDT[11]	C++17
Python	`cpg-language-python`	Python `ast`[12] (through Jep[13])	Python 3.10
Go	`cpg-language-go`	Go `parser`[14] (through JNA[15])	Go 1.20

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 21.

3.3 Delivery and usage

The following subsections detail the delivery and usage of *Codyze* for EMERALD.

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 5. 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 5.

Table 5. 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).

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

Folder / File	Description
`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 [17].

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

- Java SE 21 JDK[16].
- Source code of the *Codyze* for EMERALD (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/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 5) and are included in *Codyze*'s public GitLab repository[17].

Figure 3 illustrates how evidence extractors are used for setting up targets of evaluation in the *EMERALD UI*.

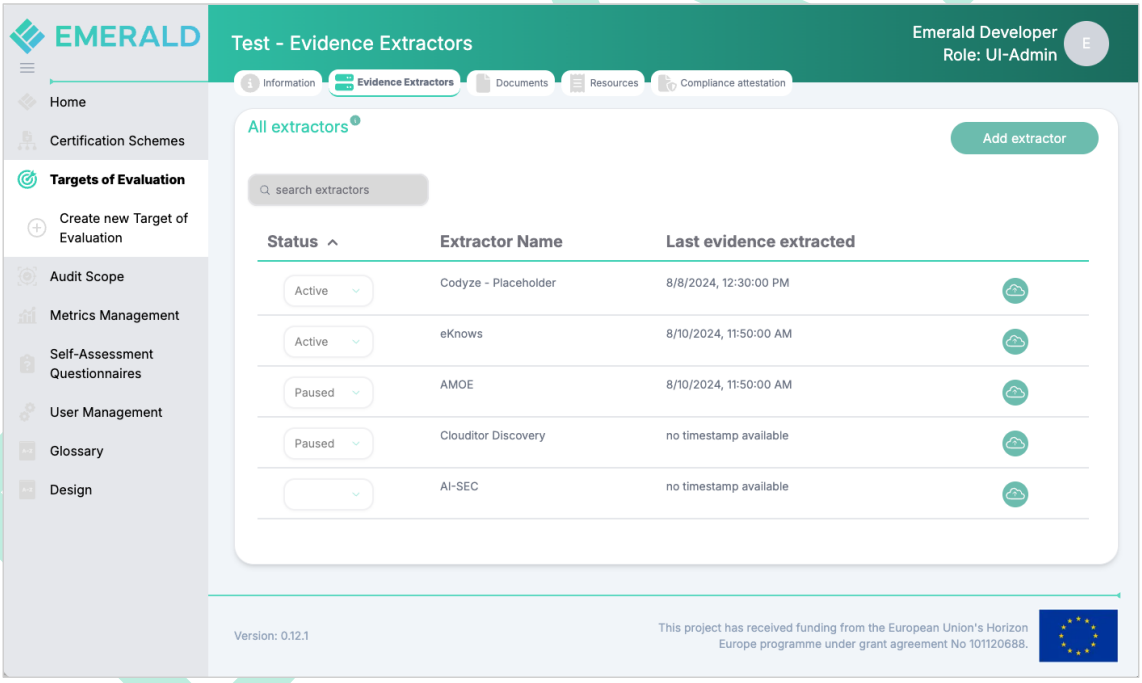

Figure 3. EMERALD UI view for setting the evidence extractor for a Target of Evaluation

Figure 4 presents the subsequent view containing configuration information for the available evidence extractors, particularly *Codyze*.

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

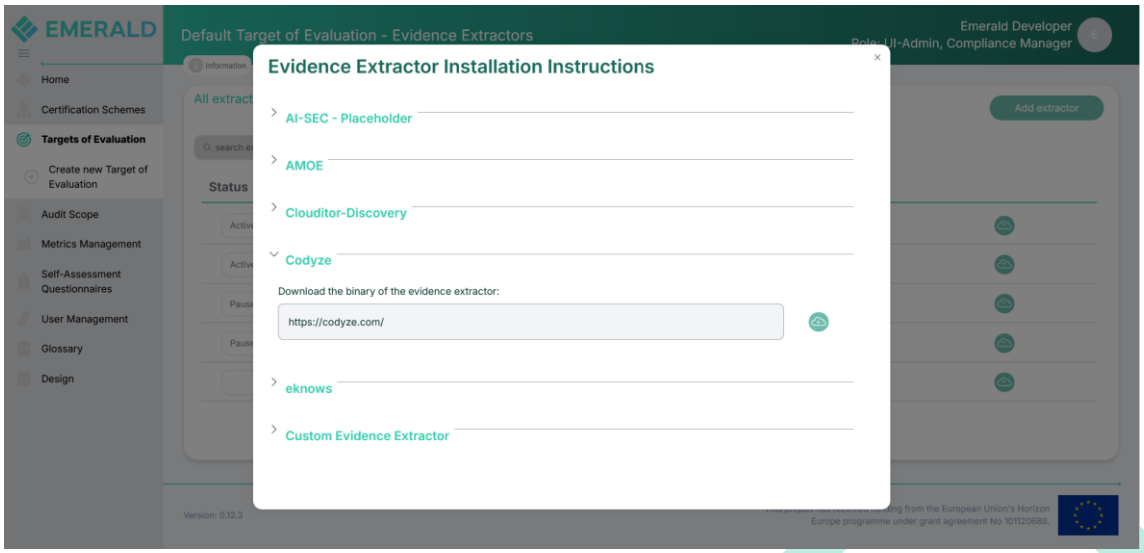

Figure 4. EMERALD UI view for the available evidence extractors and their installation instructions

Further information regarding the *EMERALD UI* can be found in D4.4 [10] and D4.6 [11].

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, i.e., 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[17] 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 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 EMERLALD component *Assessment*.

Moreover, *Codyze* for EMERALD is limited to supported programming languages and provided compliance specifications (cf. Section 3.2.2). The latter can by extended by adopting *Codyze*'s compliance specifications to user-specific cloud service and application code.

4 eknows evidence extractor (eknows-e3)

The source evidence extraction component *eknows-e3,* developed within EMERALD, builds upon relevant parts of the *eknows*[18] software analysis platform [12] [13] developed by SCCH. This platform supports rapid development of multi-language software analysis tools from pre-built modules, which are built on a technology-agnostic, generic layer and extended to meet use case-specific requirements. In turn, *eknows-e3* complements it by providing new methods and functionality for extracting knowledge from source code and delivering evidence.

4.1 Functional description

Overall purpose. To efficiently create knowledge extraction techniques delivering the required evidence to verify whether application source code complies with security requirements, we have developed the *eknows-e3* component. This component builds upon the multi-language software analysis platform *eknows*, which flexibly supports the creation of evidence extraction functions through the reuse of prefabricated parsing, analysis, and generation modules. While *eknows* provides a foundation for reverse engineering activities such as knowledge extraction, transformation, analysis, and visualization (see Figure 5), *eknows-e3* extends these capabilities by integrating different analysis modules to extract the evidence required for security-relevant assessments, for example, identifying protocols or cipher suites used in static code. Importantly, this is not limited to straightforward extraction from predefined points; *eknows-e3* additionally makes use of a resolving mechanism that can track functions or variables referenced in expected places and analyse the resolved objects in depth to uncover the necessary evidence.

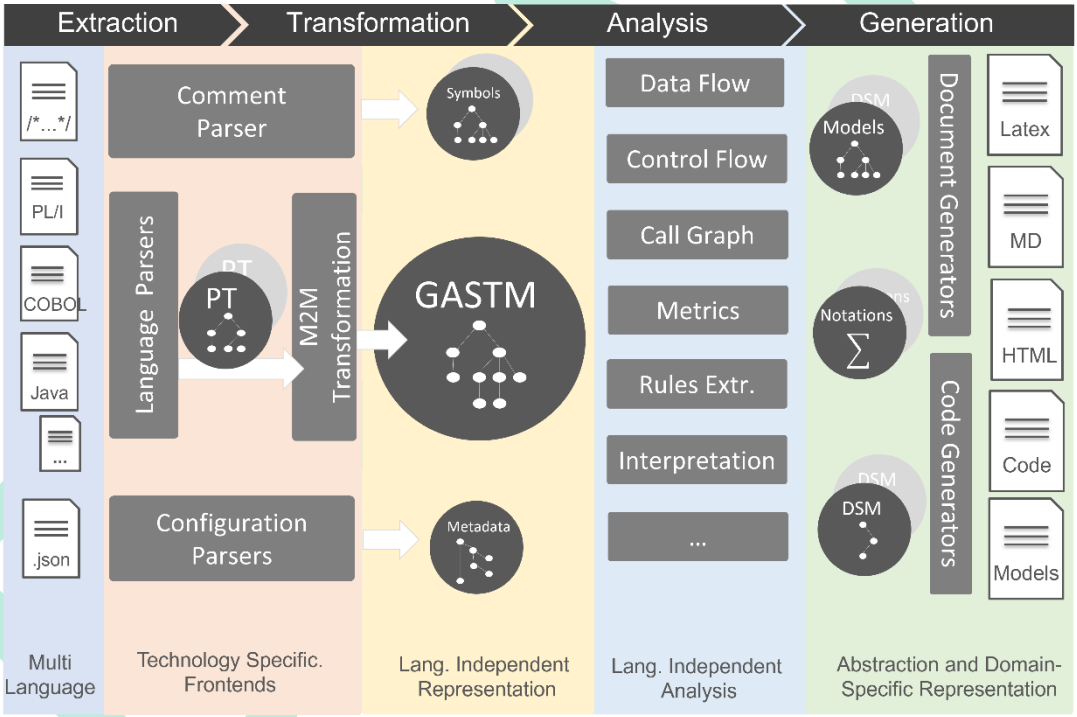

Figure 5. Reverse engineering activities supported by the software platform eknows [12]

Context and scope. While the development of the platform was originally 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. Building on this foundation, the *eknows-e3* component has been realized

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

on top of selected modules of the platform (*eknows-core*) to provide targeted functionality for EMERALD. The component makes use of extended modules for symbolic use case-specific conformity checks, fact extraction, and resolution-based tracing of functions and variables, enabling the identification of security-relevant evidence such as protocols and cipher suites from Java source code. It further applies AST-based traversal strategies to systematically analyse the source code structure, where the detection of the provider used in SSL/TLS context creation serves as the basis for determining the corresponding default and supported values. Furthermore, generation functionalities have been included to integrate this evidence into *CertGraph*, ensuring its applicability within the EMERALD framework.

At present, the scope of *eknows-e3* centres on Java, while the underlying modular architecture keeps the door open for additional programming languages and for extending the analyses to cover further security controls.

Motivation. As already said, *Codyze* and *eknows-e3* have not been technically integrated, as both can co-live in the EMERALD CaaS framework. The motivation to include *eknows-e3* 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-e3* 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 D1.4 [6] with their respective implementation progress (degree of implementation - *partially / fully /not implemented*) and a brief description of how they have been implemented are given in Table 6 to Table 10.

Table 6. 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 or by compatibility with CI/CD-Pipelines.
Status	Validated
Priority	Must
Component	*eknows-e3*
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	100% - Fully implemented
Milestone	MS3: Integrated audit suite V1 (M18)

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

Table 7. 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	Validated
Priority	Should
Component	*eknows-e3*
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	100% - Fully implemented
Milestone	MS5: Components V2 (M24)

Appropriate error messages are provided in SARIF format.

Table 8. EKNOWS.03 - Support languages required by pilots

Field	Description
Requirement ID	EKNOWS.03
Short title	Support languages required by pilots
Description	The component should be able to analyse source code written in different programming languages and should support at least Java.
Status	Validated
Priority	Must
Component	*eknows-e3*
Source	Component
Type	Technical
Related KR	KR1_EXTRACT
Related KPI	KPI 1.1
Validation acceptance criteria	The component will receive source files, should be able to process them and generate an output.
Progress	100% - Fully implemented
Milestone	MS5: Components V2 (M24)

Currently, the Java frontend is integrated.

Table 9. EKNOWS.04 - Support EMERALD evidence format

Field	Description
Requirement ID	EKNOWS.04
Short title	Support EMERALD evidence format

Field	Description
Description	The analysis of results is offered in a structured and standardized format, the EMERALD evidence format (see data model in [8]). This enables further processing and queries in other components.
Status	Validated
Priority	Must
Component	*eknows-e3*
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	100% - Fully implemented
Milestone	MS3: Integrated audit suite V1 (M18)

JSON output is generated. It includes ontology mapping and the integration of SARIF.

Table 10. 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	Validated
Priority	Must
Component	*eknows-e3*
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	100% - Fully implemented
Milestone	MS5: Components V2 (M24)

Currently, a combination of techniques is used, including AST-based parsing with a generic ASTM model, inter-procedural control flow and data flow analyses, symbolic evaluation of string values, symbol resolution across variables, fields, and calls, pattern- and rule-based extraction, and build-model inspection.

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 as the basis 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 [14] 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)[19] of the Object Management Group (OMG)) as the generic representation of the parsed source code, is another notable aspect.

4.2 Technical description

4.2.1 Prototype architecture

Figure 6 provides an overview (not fully complete) of the existing modular architecture of the *eknows* platform, which forms the technological foundation for the new *eknows-e3* component. It provides pre-built modules for (1) parsing and transforming code into a generic abstract syntax tree (AST), (2) structural and behavioural software analysis, and (3) reporting and visualization of results.

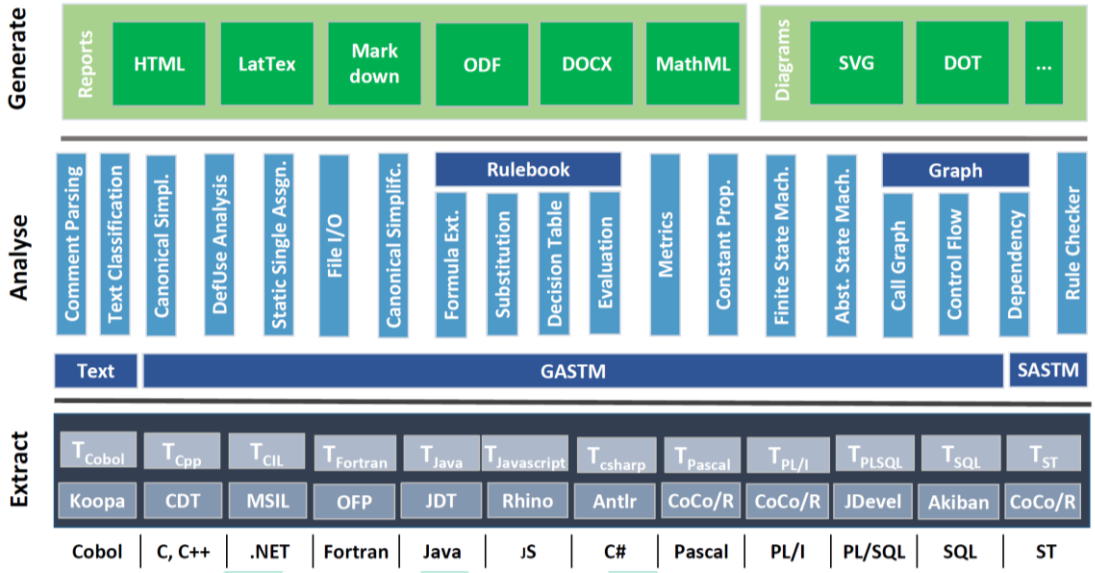

Figure 6. Overview of eknows platform building blocks [13]

The platform supports over 14 programming languages using language frontends based on parser frameworks like Eclipse JDT (Java), Eclipse CDT (C/C++), and ANTLR, or parsers generated via CoCo/R where ready-made solutions are unavailable. Parsed code is represented in a language-independent form using the OMG's Abstract Syntax Tree Metamodel (ASTM), composed of a Generic AST Metamodel (GASTM) and language-specific extensions (SASTM), allowing maximum reuse of analysis modules (such as call graph, control flow, and dependency analyses) across technology stacks. Beyond analysis, the *eknows* platform includes flexible visualization components capable of generating text, tables, charts, graphs, and structured documents in formats such as LaTeX, HTML, Markdown, and ODF. These capabilities can output both human-readable reports and structured data suitable for integration into automated workflows.

Building on this foundation, the *eknows-e3* component extends and connects selected modules of the *eknows platform* to achieve the primary goal of extracting, interpreting, and transmitting security-relevant evidence from application source code within the EMERALD framework. The component introduces three major functional parts: (1) an implementation for extracting and

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

interpreting evidence from the source code, (2) an implementation for sending extracted evidence, and (3) an integration into the CI/CD environment (see Figure 7).

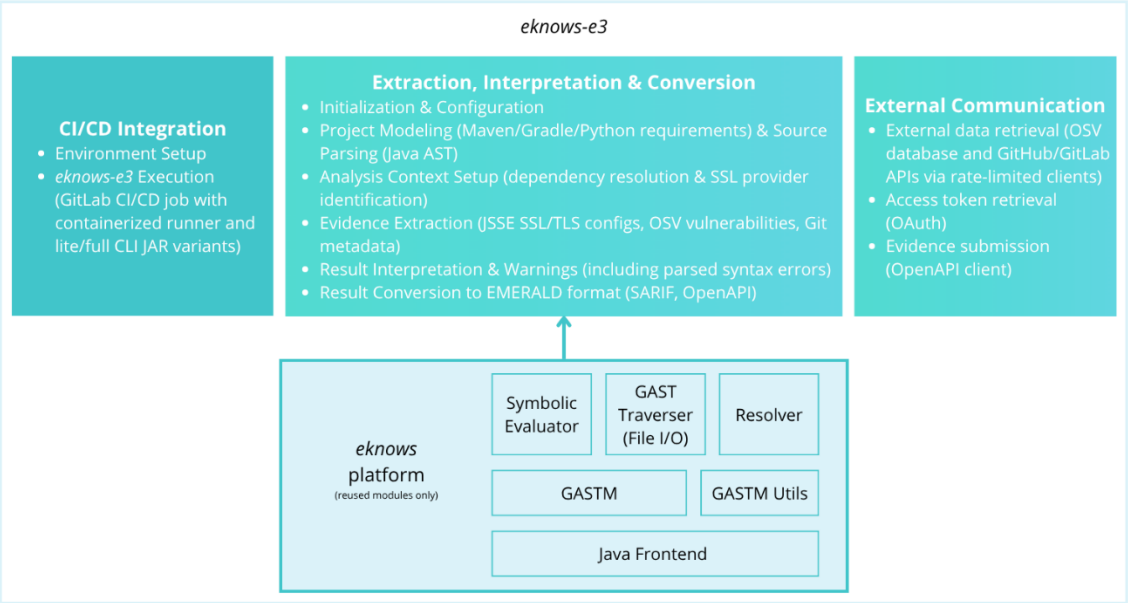

Figure 7. Overview of the eknows-e3 architecture

To support these functionalities, *eknows-e3* leverages the modular and language-independent *eknows* infrastructure while adding new layers of logic for evidence handling. At present, the developed functionality focuses on Java code analysis, with the modular architecture allowing adaptation to additional languages in the future. It integrates traversal-based and symbolic evaluation techniques with newly added logic for detecting SSL/TLS contexts, extracting protocols and cipher suites, and interpreting their configuration. In addition, *eknows-e3* incorporates mechanisms for transforming this extracted data into standardized EMERALD evidence formats and transmitting it to the *Evidence Store*.

4.2.1.1 Internal Structure of *eknows-e3*

Since the previous deliverable version, the internal structure of *eknows-e3* has gone through a notable architectural change. Originally implemented as a single Java module, the growing scope of the component made it practical to reorganize the codebase into a set of clearly separated, logically connected Maven modules. This restructuring improves maintainability, allows individual analysis capabilities to be developed independently, and enables the generation of two deployment variants adapted to different partner requirements. The resulting modular structure and its explicit dependencies are illustrated in Figure 8.

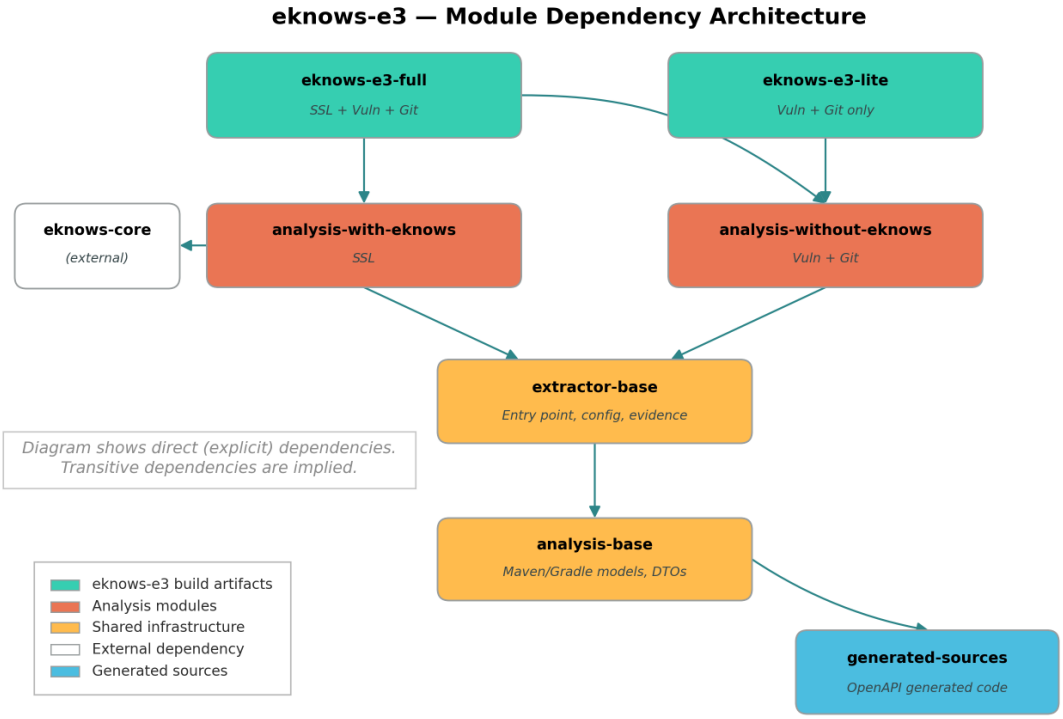

Figure 8. Overview of the restructured eknows-e3 modules and their dependencies

The restructured *eknows-e3* consists of the following modules:

- **analysis-base**: the foundational module providing shared infrastructure for all analysis activities, which includes build tool modelling for Maven and Gradle projects, the central *AnalysisModule* interface that serves as the entry point for each analysis module, and common data transfer objects, converters, and interpreter utilities.
- **analysis-with-eknows**: this module is responsible for source code analysis using the *eknows* platform infrastructure, focusing on the extraction and interpretation of SSL/TLS protocols and cipher suites from Java JSSE APIs.
- **analysis-without-eknows**: this module covers analyses that do not rely on the *eknows* platform, namely vulnerability extraction from Java and Python dependency models via the OSV database, and git-related evidence extraction from GitHub and GitLab using their official APIs.
- **extractor-base**: contains the main entry point of the *eknows-e3* application, along with configuration handling, communication settings, and shared services used across modules.
- **generated-sources**: a dedicated module holding only generated artifacts, such as OpenAPI-generated client code, consumed by other modules.
- **eknows-e3-lite/eknows-e3-full**: two packaging modules producing separate deployable JAR artifacts; the lite variant includes only git and vulnerability analysis, while the full variant additionally incorporates SSL/TLS source code analysis.

4.2.1.2 Subcomponents description

The following paragraphs detail the three main functional parts of *eknows-e3*.

Extraction and interpretation of evidence. The extraction and interpretation subsystem of *eknows-e3* forms the analytical core of the component. To initialize this analytical process, the application accepts a required set of inputs: *Evidence Store* configurations (client ID, client secret, token URL, and store URL), evidence configurations (target of evaluation ID, tool ID, and

tool name), the source directory to analyse, and the target Java version (supporting LTS versions 8, 11, 17, 21, and 25). It also accepts an optional remote repository URL (HTTPS or SSH) and a target branch; if the branch is not provided, it falls back to the repository's default or locally checked-out branch.

First, it constructs a *project model* for the underlying build tools. For Maven projects, the tool automatically creates a Maven model of the given input directory using A*pache Maven*[20]. This model captures the hierarchical structure of the project, aggregates modules, and records their associated dependencies. For Gradle projects, the component employs a dual-strategy approach: it first runs a dynamic analysis using the *Gradle Tooling API*[21], which automatically resolves all dependencies. If this dynamic approach fails, which can occur if a legacy Gradle version is incompatible with the modern Java version (17+) running the *eknows-e3* instance, the tool falls back to a static analysis using the *OpenRewrite Gradle plugin*[22] to parse the build files and manually resolve dependencies. In parallel, using functionality provided by the *eknows* platform, a GAST model of the project's source files is generated. During this phase, a *JavaFrontend* parser converts the source code into compilation units, specifically identifying and saving files that contain syntax errors. These error files are then reported to the *Evidence Store* in raw SARIF format. The resulting GAST compilation units represent individual files as the highest-level entities in the GAST metamodel, which serves as the structural foundation for the subsequent source code analysis. Independently, within the constructed build project model, properties defined in parent Maven or Gradle modules are resolved and propagated to child modules. This ensures the accurate determination of dependency names, versions, and configurations. This resolution process also forms the basis for identifying security-relevant dependencies and potential vulnerabilities later in the analysis pipeline.

An important focus of this module is the identification of security providers used in the examined project. Using the Maven or Gradle model, the component checks whether the *BouncyCastle* library is included as a dependency. This step is essential because *eknows-e3* currently considers two main providers for SSL/TLS evidence extraction — *SunJSSE* and *BouncyCastleJSSE*. These providers can define different default sets of cipher suites and protocols that are used if no explicit configurations exist. When *BouncyCastle* is detected, *eknows-e3* retrieves the exact version of the library from Maven Central, loads the corresponding classes, and dynamically instantiates an *SSLContext* using the *BCJSSE* provider.

Subsequently, *eknows-e3* performs an *AST-based traversal* of the parsed source code to detect code structures related to SSL context creation and configuration. The traversal systematically identifies points in the source code that correspond to SSL context instantiation and configuration logic. Using a *resolution mechanism*, variable and function references are resolved so that even indirect provider definitions can be traced back to their concrete origins.

The following stage represents the main evidence extraction process, focusing on *protocols* and *cipher suites*. In this version, the emphasis is on extracting SSL/TLS configurations implemented using the standard *Java Secure Socket Extension* (JSSE) library. For this, *eknows-e3* uses the symbolic evaluator available in the *eknows* platform, which extracts symbolic (string) values from expressions, constructors, and function calls. The extraction process constructs a *hierarchical tree* that connects these objects and models SSL-active sections of the application (e.g., socket creation and configuration). This tree-based representation provides a

[20] https://maven.apache.org/
[21] https://gradle.org/
[22] https://plugins.gradle.org/plugin/org.openrewrite.rewrite

comprehensive view of how SSL contexts are configured and connected, enabling later interpretation of misconfigurations or ambiguities.

After extraction, the *interpreter module* processes the tree-based results to produce semantically meaningful evidence entities, mapping each finding to its module and activation context for traceability. For protocols, the interpreter derives protocol family and version (e.g., TLS and 1.3) and detects usage contexts such as multiple simultaneous settings that may introduce ambiguity, flagging these cases with standardized warnings for later reporting in SARIF format (JSON data in the raw evidence field) and in EMERALD format (following the ontology terms). In addition, all explicitly configured protocol entries are treated as enabled, and when a configuration sets exactly one protocol in the array, that protocol is considered enforced. For cipher suites, the interpreter applies a dedicated parsing stage that normalizes naming variants and, using pattern-based rules, derives the constituent elements expected by the EMERALD model: key exchange mechanism, authentication mechanism, cipher, and MAC, with special handling for pseudo suites and export indicators. The interpreter also classifies malformed, unknown, unsupported, or pseudo cipher suites, and raises warnings for multiple or conflicting settings, default (unspecified) configurations, and unused definitions in alignment with evidence reporting best practices. In cases where no protocol or cipher suite is explicitly configured at an SSL-active location, the interpreter links the location to its resolved *SSLContext* and infers the effective configuration from the provider defaults (SunJSSE[23] or BCJSSE[24]) corresponding to the detected provider version and the specified Java version of the analysed code, ensuring that implicit defaults are represented in the resulting evidence.

Beyond source code evaluation, the subsystem performs independent vulnerability analysis by extracting direct dependencies and their first-degree indirect dependencies from the constructed Maven or Gradle project models. In addition, as requested by a pilot partner, Python ecosystems are supported through dependency extraction from *requirements.txt* files using an *ANTLR* parser[25]. Identified dependencies are checked against the *Google OSV (Open Source Vulnerability)* database[26] via its API. This database was selected because it is the largest open-source vulnerability database available, covering approximately 300,000 advisories across more than 30 ecosystems. To improve performance, this extraction runs concurrently with exponential backoff for retrying failed requests.

At the same time, the component collects Git-related security evidence from GitHub and GitLab using their official APIs, authenticated through a repository access token (personal or project-level). The extracted information includes repository security policies (such as enforced developer sign-off, signed commits, approved author policies, and mandatory two-reviewer approval), commit verification statuses, and pull request metrics such as the number of reviewers on merged pull requests. For local Git repositories, commit messages are also analysed to verify developer sign-off. Since the volume of API requests grows with repository size, the extraction runs concurrently to speed up the process. However, as Git platform APIs limit the number of requests and concurrent calls, the module implements a platform-specific rate limiter: for GitHub, a fixed cap of 10 requests per second is applied, while for GitLab, the remaining rate limit for the current minute is retrieved dynamically, converted to a per-second rate, and scaled down by a factor of 0.8 to provide a safety margin. In both cases, this is combined with exponential backoff and retry logic for failed calls. This approach achieves a

[23] https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html#SunJSSE
[24] https://www.bouncycastle.org/
[25] https://www.antlr.org/
[26] https://osv.dev/

speedup of approximately 3.8x to 4.2x over sequential execution, reaching up to 98.5% of the theoretical maximum throughput allowed by the rate limits.

The interpreted information is collected into result objects, which summarize relevant data and associated warnings. These are then converted into *Resource* objects—data structures generated from the EMERALD OpenAPI specification through the *openapi-generator-maven-plugin*[27]—and enriched with corresponding SARIF structures. The SARIF representations, created using the *sarif4k* library[28], provide standardized reporting of findings, supporting further integration and analysis by other EMERALD components.

Sending evidence. Once the evidence objects are prepared, the sending subsystem of *eknows-e3* uses the *ScribeJava* OAuth client library[29] and the generated OpenAPI client to authenticate and transmit the results to the EMERALD *Evidence Store*. The OpenAPI-based communication layer is generated automatically using the *openapi-generator-maven-plugin*, which produces the required client classes directly from the EMERALD-provided OpenAPI specification. The sending process acquires an OAuth access token from the specified authorization server, attaches it to outgoing requests, and submits the *Resource* and SARIF data to the *Evidence Store*.

Integration into the CI/CD environment. The *eknows-e3* component can operate as a standalone executable binary (JAR) and is designed to integrate smoothly into continuous integration and deployment pipelines. To support varied deployment contexts, the build system generates two distinct JAR artifacts: *eknows-e3-lite*, which performs only the vulnerability and Git analyses without relying on the *eknows-core*, and *eknows-e3-full*, which executes the complete suite of analyses including the *eknows*-based SSL/TLS source code evaluation. Within EMERALD, integration has been implemented using *GitLab CI/CD*, where a dedicated component definition allows flexible configuration of parameters, such as OAuth credentials, Evidence Extractor URL, signing key, target ID, and source directory path. The created CI/CD component executes the *eknows-e3* binary on the specified source directory, triggering analysis, evidence extraction, interpretation, and automated submission of results to the *Evidence Store*. The flexible interface allows configuration through environment variables, enabling seamless integration into diverse CI/CD setups and deployment contexts.

4.2.2 Technical specifications

The *eknows* platform, on which *eknows-e3* builds, is a Java-based framework comprising over 350K source lines of code (SLOC). It reuses modular analysis and generation components with language-specific parsers (see Table 11) integrated through compiler-generator tools such as Eclipse CDT[30], JDT[31], and ANTLR [32]. The system maintains a generic representation of parsed content based on the ASTM metamodel and manages dependencies through Maven[33].

Table 11. Supported programming languages by eknows

Language	Parser	Supported version
Adele	CoCo/R[34]	n.a.
B&R (Bernecker + Rainer)	CoCo/R	n.a.

[27] https://openapi-generator.tech/docs/plugins/
[28] https://github.com/detekt/sarif4k
[29] https://github.com/scribejava/scribejava
[30] https://projects.eclipse.org/projects/tools.cdt
[31] https://www.eclipse.org/jdt
[32] https://www.antlr.org/
[33] https://maven.apache.org/
[34] https://ssw.jku.at/Research/Projects/Coco

Language	Parser	Supported version
C, C++	Eclipse CDT[30]	C++17
C#	ANTLR[32]	C# 7.0
CIL/.NET	n.a.	.NET 4.5
COBOL	Koopa[35]	COBOL 85
Codesys/Bachmann	Xtext[36]	3
Fortran	Open Fortran[37]	77/2003
JCL	CoCo/R	JCL z/OS 2.2
Java	Eclipse JDT[31]	Java 8
Javascript	Mozilla Rhino[38]	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
PL/SQL	JDeveloper/Akiban[39]	9.1
Python	ANTLR	Python 3.6
Sigmatek	ANTLR	n.a.
SQL	Akiban	MySQL 5.6

The *eknows-e3* component, composed of over 10K source lines of code, inherits this architecture but extends it with additional modules for source-based security evidence extraction and automatic evidence delivery within the EMERALD infrastructure. It thus represents a specialized evolution of the *eknows* analysis framework tailored to support EMERALD's trustworthiness assurance workflows.

Currently, focused on Java code analysis, *eknows-e3* consists of an executable binary distribution and runs stand-alone. Like *Codyze*, it is executed on source code of cloud applications and services and is integrated into CI/CD pipelines by using the binary distribution. Currently, a CLI is provided, and a REST interface can be provided in the future, if needed.

4.3 Delivery and usage

The following subsections detail the delivery and usage of *eknows-e3*.

4.3.1 Package information

The *eknows-e3* is developed as a Java application with the support of Maven[40] as build tool. Following an architectural restructuring, the codebase has been reorganized from a single Java module into a set of clearly separated, logically connected Maven modules. Table 12 shows the updated structure of the Gitlab repository[41] and its contents.

Table 12. Overview and description of package structure for eknows-e3

Folder	Description
`analysis-base/`	The foundational Maven module providing shared infrastructure for all analysis activities. Includes build tool modelling for Maven and Gradle

[35] https://github.com/krisds/koopa
[36] https://eclipse.dev/Xtext/
[37] https://github.com/OpenFortranProject/open-fortran-parser
[38] https://github.com/mozilla/rhino
[39] https://github.com/brunoribeiro/sql-parser
[40] https://maven.apache.org/download.cgi
[41] https://git.code.tecnalia.dev/emerald/public/components/eknows

	projects, the central `AnalysisModule` interface, and common data transfer objects, converters, and interpreter utilities.
`analysis-with-eknows/`	Maven module responsible for source code analysis using the *eknows* platform infrastructure. Focuses on extraction and interpretation of SSL/TLS protocols and cipher suites from Java JSSE APIs.
`analysis-without-eknows/`	Maven module covering analyses that do not rely on the *eknows* platform: vulnerability extraction from Java and Python dependency models via the OSV database, and Git-related evidence extraction from GitHub and GitLab using their official APIs.
`ci/`	The directory with CI/CD pipelines for development. Includes pipelines for testing and automatic pushing the *eknows-e3* binary file to the Artifactory.
`eknows-e3-full/`	Packaging module producing the full deployable JAR artifact. Incorporates all analysis capabilities, including Git and vulnerability analysis as well as *eknows*-based SSL/TLS source code evaluation.
`eknows-e3-lite/`	Packaging module producing the lite deployable JAR artifact. Includes only Git and vulnerability analysis, without relying on the *eknows* platform or SSL/TLS source code evaluation.
`eknows/`	The prebuilt *eknows* binaries should be placed here. In addition, installation scripts are provided in this folder.
`extractor-base/`	Maven module containing the main entry point of the *eknows-e3* application, along with configuration handling, communication settings, and shared services used across all modules.
`generated-sources/`	A dedicated Maven module holding only generated artifacts, such as OpenAPI-generated client code consumed by other modules.
`templates/`	CI/CD component directory with configuration file.
`.gitignore`	Git ignore configuration for the repository.
`.gitlab-ci.yml`	CI/CD pipeline file, which runs pipelines in `ci/` directory.
`LICENSE`	License text (Apache License, Version 2.0).
`README.md`	Compact guide on how to build and use the extractor.
`pom.xml`	Root Maven POM file defining the multi-module project structure and shared build configuration for all submodules.

4.3.2 Installation

Requirements:

- Java 17 JDK[42].
- Maven[40].
- Source code of the *eknows-e3* (see Section 4.3.5).

The following steps are required to build *eknows-e3*:

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

[42] https://adoptium.net/de/temurin/releases/?version=17&package=jdk

c. Run `install.cmd` or `./install` within the *eknows* folder

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

3. The built extractors can be found at:
   a. `eknows-e3-lite/target/eknows-e3-lite-<version>-jar-with-dependencies.jar`: includes only Git and vulnerability analysis.
   b. `eknows-e3-full/target/eknows-e3-full-<version>-jar-with-dependencies.jar`: includes the complete analysis suite, additionally incorporating SSL/TLS source code evaluation.

4.3.3 Instructions for use

The *eknows-e3* component can be run from the command line by invoking:

```
java -jar eknows-e3-full/target/eknows-e3-full-<version>-jar-with-
dependencies.jar -sd <dir to analyse> -jv <java version of analysed dir>
```

This will extract the evidence, interpret the information, and send it to the *Evidence Store*. The current version of the extractor also fetches the sent evidence from *Evidence Store* and prints them out. (Note: Also, the `-lite` variant can be used, depending on the required analysis scope – see Section 4.3.2.)

Configuration is provided through environment variables. The variables are listed in Table 13.

Table 13. Environment variables for configuration of eknows-e3

Environment Variable	Description
clouditor_clientId	OAuth client ID for the *Evidence Store*
clouditor_clientSecret	OAuth client secret for the *Evidence Store*
clouditor_tokenUrl	Token endpoint URL of the authorization server
clouditor_storeUrl	Base URL of the *Evidence Store*
evidence_targetOfEvaluationId	Target of evaluation ID
evidence_toolId	Tool identifier
evidence_toolName	Human-readable tool name

The CLI arguments shown in Table 14 are supported.

Table 14. CLI arguments for configuration of eknows-e3

Argument	Short	Required	Description
--srcDir	-sd	Yes	Source directory to analyse
--javaVersion	-jv	Yes	Java LTS version of the analysed code (8, 11, 17, 21, 25)
--remote	-r	No	Remote repository URL (HTTPS or SSH) for Git evidence extraction
--branch	-b	No	Target branch to analyse; falls back to the repository's default branch if not specified

Figure 3 illustrates how the source evidence extractors will be used for setting up targets of evaluation in the *EMERALD UI*. Figure 9*Figure* 4 presents the subsequent view containing configuration information for the available evidence extractors, particularly e*know-e3*. Please refer to D4.4 [10] and D4.6 [11] for further visualizations of the *EMERALD UI*.

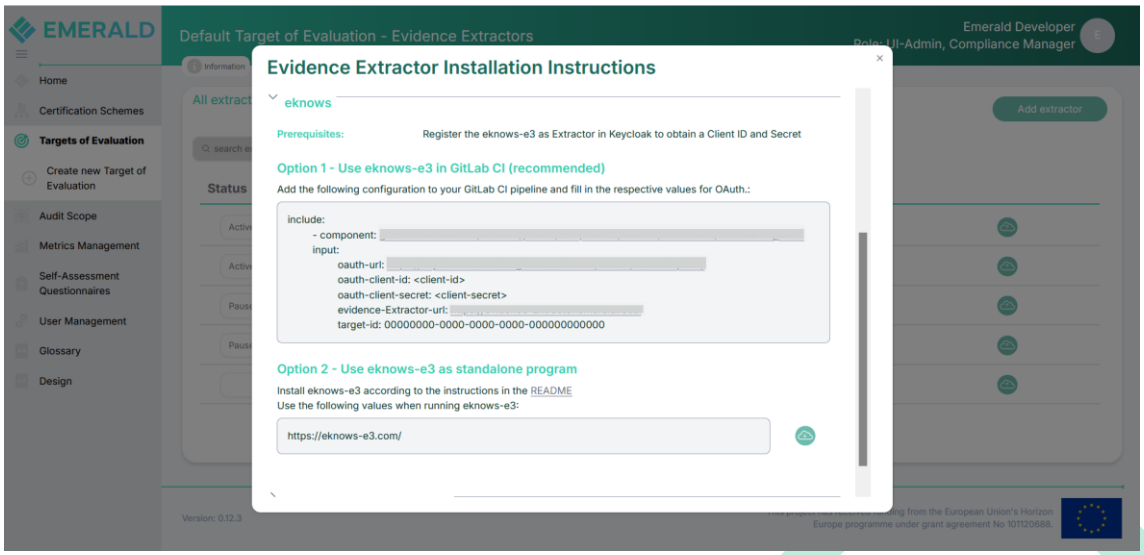

Figure 9. EMERALD UI view for the available evidence extractors and their installation instructions

In addition to command-line execution, the *eknows-e3* component can also be integrated and triggered directly within a GitLab CI/CD pipeline. This allows to use automated evidence extraction as part of continuous integration workflows. The component can be included and configured as follows:

```
include:
      - component:
      "git.code.tecnalia.dev/emerald/public/components/eknows/eknows-
      e3/eknows-e3@main"
        inputs:
           oauth-url: "http://localhost:8080/v1/auth/token"
           oauth-client-id: "confirmate"
           oauth-client-secret: "confirmate"
           evidence-collector-url: "http://localhost:8080/"
           target-id: "00000000-0000-0000-0000-000000000000"
           src-dir-path: "testdir"
           java-version: "17"
           remote: "$CI_SERVER_URL/$CI_PROJECT_PATH.git"
```

Versioning is supported for the *eknows-e3* CI/CD component, allowing users to reference specific tagged releases within their pipeline configurations. The first versioned release, **@v2**, has been available for an extended period. With the publication of this deliverable, **@v3** will be released, introducing the latest updates and improvements described in this document.

Also, a demo repository, named *eknows-e3 – component demo*[43], is available to showcase how the component can be included and executed within a GitLab pipeline. This repository analyses a target repository, which is also used internally for integration testing of the *eknows-e3* component. Additionally, when changes are merged into the main branch of the *eknows-e3* repository, the pipeline in the demo repository is automatically triggered to verify that the CI/CD component, in combination with the updated Artifactory image of *eknows-e3*, functions without errors (please note that this works only within the internal repository).

[43] https://git.code.tecnalia.dev/emerald/private/components/eknows/eknows-e3-demo-repository

4.3.4 Licensing information

The licensing is split into two parts:

1. The *eknows-e3*, which is developed in the context of EMERALD, is licensed under Apache 2.0[44] 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-e3* is available from the public EMERALD GitLab repository[45] hosted by TECNALIA. The repository hosts 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[46].

4.4 Limitations and future work

Also, current developments in the field of Large Language Models (LLMs) open new ways to analyse code. Although it looks quite promising to analyse source code by LLMs, this approach is inherently limited by the probabilistic nature of LLMs, and their outputs lack formal guarantees. On the other hand, static code analysis can provide exact results but also has its limitations. This means that all rules and checks have to be written by hand. Therefore, future work will also address ways to combine the strengths of LLMs and static code analysis. Our current experiments focus on the integration of static code analysis tools into coding agents. The coding agent receives source code and an initial prompt as input. Subsequently, control is handed over to the coding agent, which can then plan and decide on further steps, including accessing static code analysis tools as needed or spawning sub agents to handle smaller tasks. As part of this effort, we are evaluating coding agents against the dataset PrimeVul[47], which contains labelled pairs of vulnerable and benign code versions. The evaluation will consider various experimental variations, including different underlying LLM models and a comparison of performance with and without static code analysis. It will also examine the conversation flow between the agents to gain insights into how the agents interact with each other and with the provided tools. The results of this study are planned to be published in a scientific paper.

Looking further ahead, several open questions and challenges remain for *eknows-e3*. Language support beyond Java and Python could grow based on community demand and pilot needs. Full Git analysis sometimes requires elevated administrative privileges, which can be difficult to obtain in enterprise environments. Finally, the growing number of LLM-based code analysis tools represents a competitive challenge, reinforcing the value of *eknows-e3* as the only standalone tool that combines SSL/TLS code analysis, dependency vulnerability scanning, and Git integrity metrics in a single, deterministic output built for continuous certification workflows.

[44] http://www.apache.org/licenses/LICENSE-2.0
[45] https://git.code.tecnalia.dev/emerald/public/components/eknows
[46] https://git.code.tecnalia.dev/emerald/private/components/eknows/eknows-core-for-emerald [internal use only - authentication required]
[47] https://github.com/DLVulDet/PrimeVul

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 final output of Task 2.2, we presented the technical report about the design, architecture, and implementation 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-e3*. The components, based on some background works, have been integrated with other EMERALD components and satisfy their respective requirements. To reflect all changes and refinements, an updated version of this deliverable (v1.1) was provided at the end of WP2. Future work will also deal with incorporating the feedback of pilots' evaluation regarding the needs of business requirements of the selected security controls developed in WP5.

6 References

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

[2] EMERALD Consortium, “D2.5 AMOE – v2,” 2026.

[3] EMERALD Consortium, “D2.7 ML model certification – v2,” 2026.

[4] EMERALD Consortium, “D2.9 Runtime evidence extractor – v2,” 2026.

[5] EMERALD Consortium, “D2.2. Source evidence extractor - v1,” 2024.

[6] EMERALD Consortium, “D1.4 EMERALD solution architecture - v2,” 2025.

[7] EMERALD Consortium, “D3.4 Evidence Assessment and Certification-Implementation - v2,” 2025.

[8] EMERALD Consortium, “D1.2 Data modelling and interaction mechanisms - v2,” 2025.

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

[10] EMERALD Consortium, “D4.4 User interaction and user experience concept – v2,” 2025.

[11] EMERALD Consortium, “D4.6 EMERALD UI - v2,” 2026.

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

[13] 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.

[14] 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