Codyze – Ein Schritt zur automatisierten Nachweisführung

21.06.2022

Alexander Küchler

Fraunhofer AISEC

Projektpartner

- **Bundesamt für Sicherheit in der Informationstechnik (BSI)**
  - Projektfinanzierung

- **TÜV Informationstechnik GmbH (TÜViT)**
  - Pilotierungsteilnehmer

TÜViT®
TÜV NORD GROUP

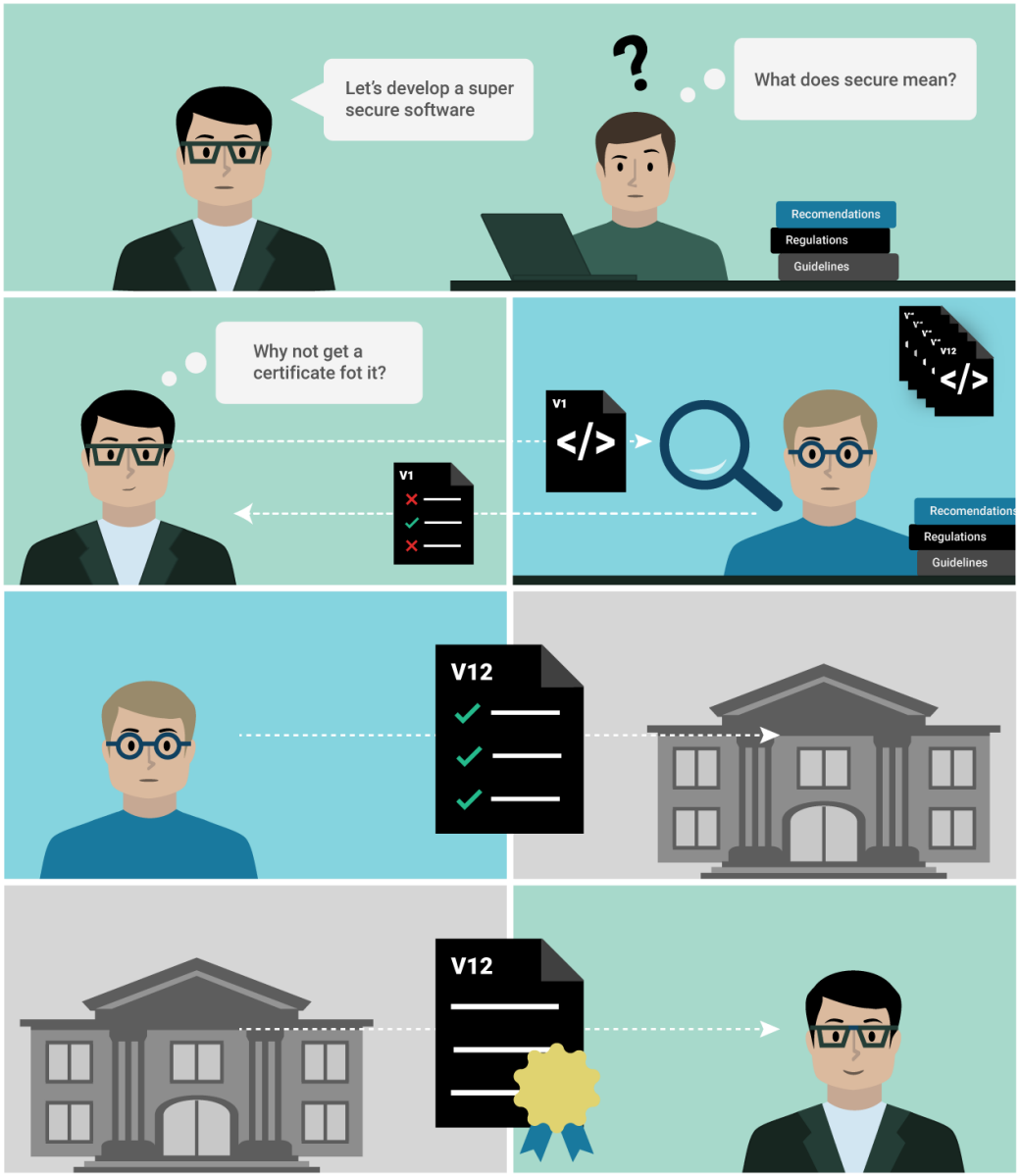
Fraunhofer AISEC

Motivation

- Sichere Software ist wichtig!
- Sichere Software ist schwer zu schreiben!
- Zertifizierung häufig wünschenswert oder sogar gefordert

Stakeholder

- Entwickler
- Auditor
- Evaluierungsstelle



Fraunhofer AISEC

Motivation
Standards

- Zunehmende Anzahl von Regularien im Bereich IT-Sicherheit: GDPR, KRITIS / IT-Sig 2.0, EU Cybersecurity Act, ISO 62443, ISO 27070, BSI C5, Common Criteria, VS-Zulassung
  - Meist high-level und Service-agnostisch z.B. Referenzieren den State-of-the-Art
  - Nachweis (z.B. bei Zertifizierung) erfolgt für eine konkrete Implementierung
- Technische Definition des State-of-the-Art nötig
  - ➔ Interpretation durch Experten nötig
- Handlungsempfehlungen z.B. Technische Richtlinien sind typischerweise in Prosa geschrieben

➔ **Aufwendige Überprüfung, Interpretationsspielraum**

Fraunhofer AISEC

Motivation
Softwareentwicklung

- Immer kürzere Entwicklungszyklen
- Zunehmender Druck auf Entwickler, die Anforderungen zu erfüllen
- Steigende Komplexität der Software
  - Abhängigkeiten zu Third-Party-Bibliotheken
  - Erhöhte Anzahl von Lines-of-Code

→ **Begünstigt Fehler in der Entwicklung**

Motivation

Anwendungsfall: Kryptografische Bibliotheken

- Richtiger Einsatz von Kryptografischen Bibliotheken ist schwer [1]
  - Unintuitive Dokumentation
  - Keine intuitive API
  - Unsichere Defaults
  - Fehlendes Hintergrundwissen der Entwickler
- Stack Overflow als beliebte Anlaufstelle für Entwickler [2]
  - Oft funktionaler Code
  - Schnell in der Entwicklung
  - Aber: häufig unsicherer Code

Quellen:
[1] Acar et al. Comparing the Usability of Cryptographic APIs. IEEE S&P 2017
[2] Acar et al. How Internet Resources Might Be Helping You Develop Faster but Less Securely. IEEE S&P 2017

Lösungsansatz: Automatisierte Analyse

- Ziel: Lücke zwischen Expertenwissen und Entwicklern reduzieren
- **Modellierung von Anforderungen in (erweiterbaren) *Regeln***
- **Unterstützung von Entwicklern**
  - Integration in Entwicklungsumgebungen
  - Integration in Continuous Integration Pipelines
- **Unterstützung von Auditoren**
  - Funktioniert mit nicht kompilierbarem Code

Komponenten: CPG, MARK & Codyze

- Code Property Graph (CPG)
  - Repräsentiert Source Code verschiedener Sprachen
- MARK (Modellierungssprache für Anforderungen und Richtlinien der Kryptografie)
  - Formale und maschinenlesbare Modellierungssprache
  - Anforderungen an Objekte in Source-Code
- Codyze
  - Statisches Code Analyse Werkzeug
  - Liefert Nachweis über die korrekte Implementierung basierend auf einem Graphen
  - Benutzt MARK, um die Regeln vom Analyse Werkzeug zu entkoppeln

Link: https://www.codyze.io

Komponenten: CPG, MARK & Codyze

Regulatorien

```
rule _2_1_01_Modes {
  using Botan.Cipher_Mode as cm
  ensure
    _split(cm.algorithm, "/", 1) in ["CCM",
"GCM", "CBC", "CTR"]
  onfail WrongMode
}
```

MARK

Codyze

- Violation: NoRandomIV
- Violation: WrongMode
- Correct

Source Code

CPG

Fraunhofer AISEC

Code Property Graph

Repräsentation von Code für Analyse

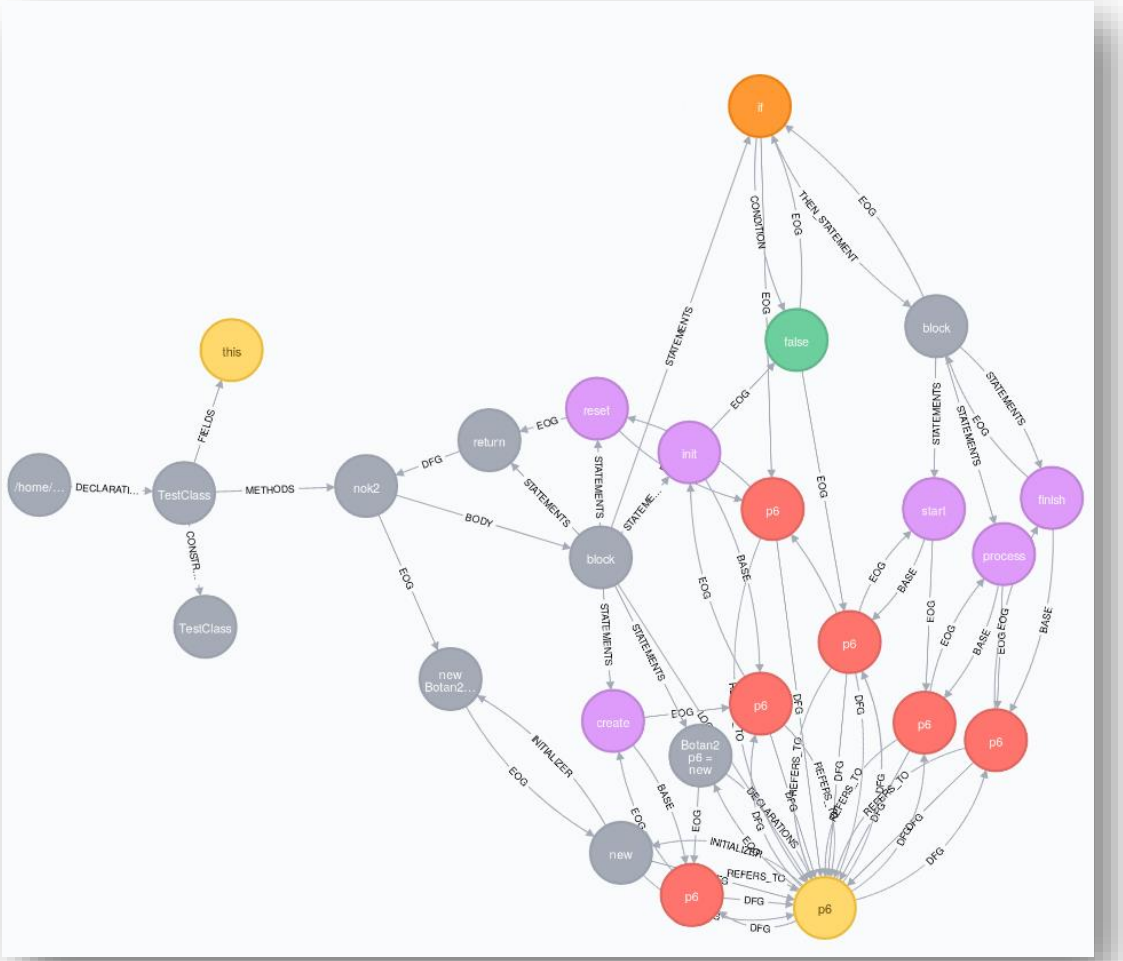
- Code Property Graph verknüpft mehrere Graphen: AST, CFG, CG
- CPG enthält alle Informationen aus dem Source Code
- „Labeled Multigraph“: Knoten und Kanten haben ein *Label* und eine Menge von *Properties*
- Durch zusätzliche Analysen können weitere Kanten hinzugefügt werden: Datenflüsse, Auswertungsreihenfolge
- Möglich auch für unvollständigen Code (bspw. bei fehlenden Includes)
- Unterstützte Programmiersprachen: Java, C/C++, Python, Go, TypeScript, LLVM

Eigene Implementierung mit vielen weiteren Features: https://github.com/Fraunhofer-AISEC/cpg

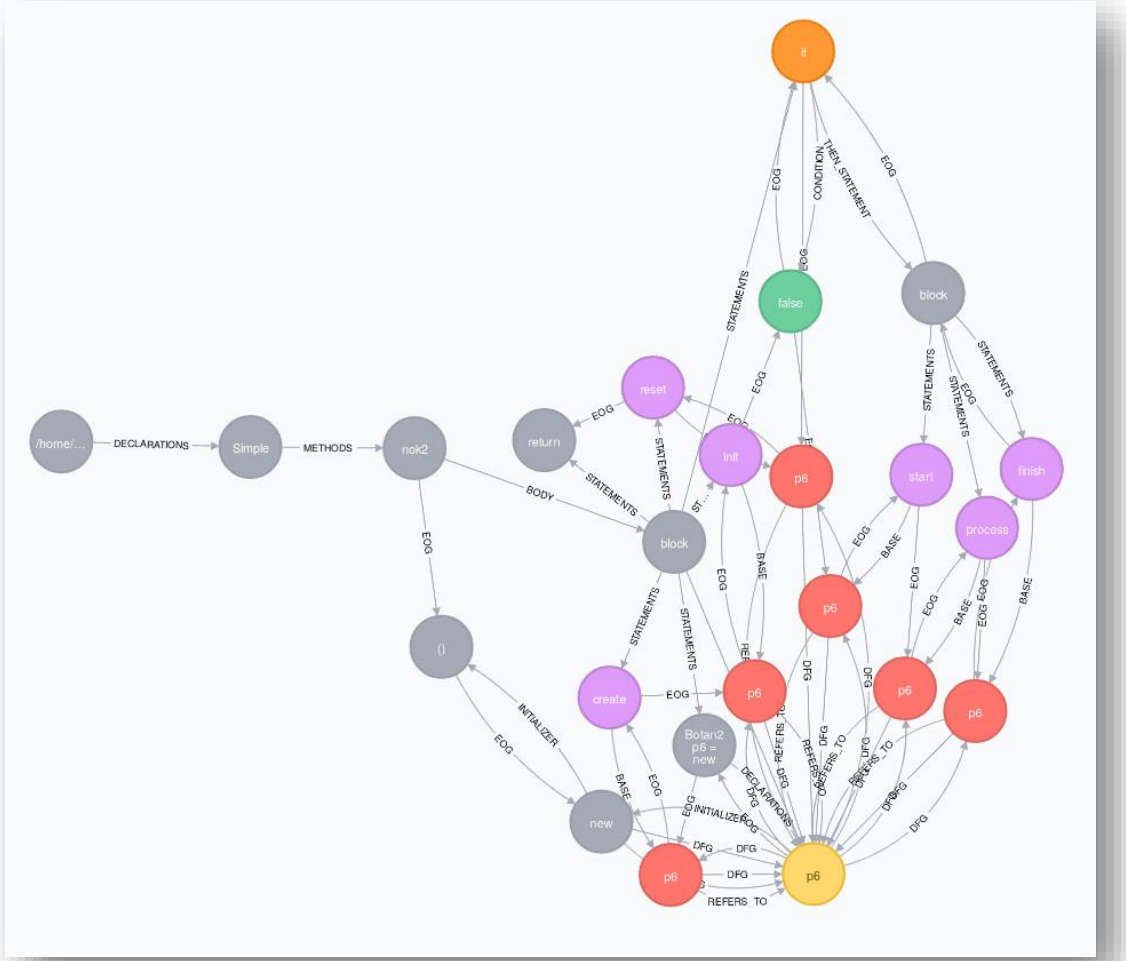
Test it, it‘s free! **Star it on github!** ★

Code Property Graph

Sprachenunabhängigkeit



Java



C++

Fraunhofer AISEC

MARK
Modell und Regeln

- Notation von Modell und Regeln
  - Modell einer Bibliothek ➔ Definiert Operatoren und Variablen
  - Regeln für das Modell ➔ Definiert die Anforderungen
- Einfache Operationen: Strings aufteilen, Konstanten/Literale auflösen, Datenflüsse verfolgen
- Was kann geprüft werden?
  - Reihenfolge von Funktionsaufrufen
  - Anforderungen an Parameter
- Einfach zu schreiben
- Auf kryptografische APIs und häufige Fehler spezialisiert
- Aber: Erstellung der Regeln erfordert Expertenwissen

MARK
Modell und Regeln

```
entity Cipher_Mode isa Cipher {
    var nonce : uint8_t;
    var nonce_length : std::size_t;
    var iv : Botan::InitializationVector;
    var algorithm : std::string;

    op create() {
        Botan::get_cipher_mode(algorithm, _);
    }

    op init() {
        Botan::set_key(_);
        Botan::set_key(_, _);
    }

    op start() {
        Botan::start(iv);
        forbidden Botan::start_msg(*);
    }

    […]

}
```

```
rule UseOfBotan_CipherMode {
    using Order as cm
    ensure
        order cm.create(), cm.init(), cm.start()
    onfail WrongUseOfBotan_CipherMode
}

rule BlockCiphers {
    using Botan::Cipher_Mode as cm
    ensure
        _split(cm.algorithm, "/", 0) in [ "AES" ]
    onfail WrongBlockCipher
}

rule UseRandomIV {
    using Botan::Cipher_Mode as cm,
          Botan::AutoSeededRNG as rng
    when _split(cm.algorithm, "/", 1) == "CBC" &&
          cm.direction == Cipher_Dir::ENCRYPTION
    ensure
         _receives_value_from(cm.iv, rng.random)
    onfail NoRandomIV
}
```

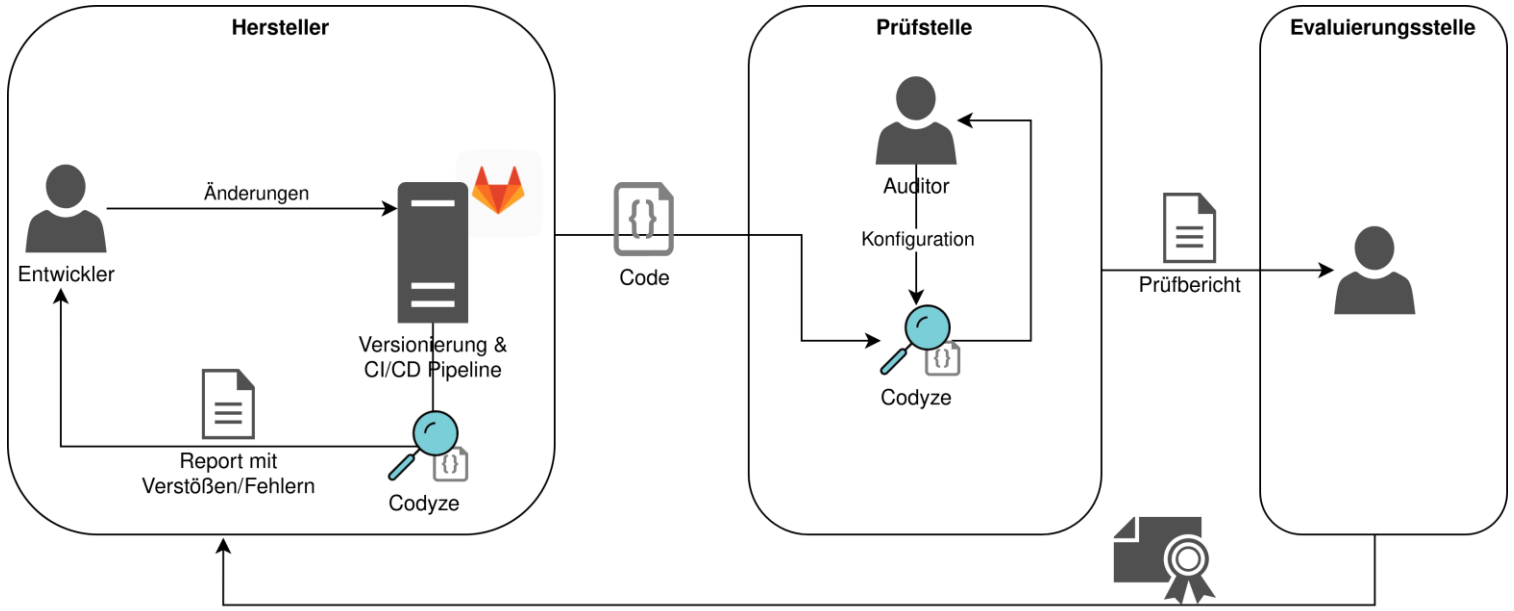
Codyze
Ziele & Umsetzung

- Automatisierte Prüfung von Compliance des Source Codes anhand eines Standards
  - Der Standard wurde durch MARK-Regeln für eine konkrete Bibliothek modelliert
  - Der Source Code wurde auf den CPG abgebildet
- Übersetzung der Regeln in Anfragen an den CPG
- Prüft, dass für alle Knoten im CPG alle MARK Regeln eingehalten werden
- Whitelisting: Alles was nicht explizit erlaubt ist, ist falsch
- Analysen:
  - Typestate Analyse
  - Wertebelegungen von Variablen
  - Datenflüsse

Codyze

Ziel: Integration in Prozesse

- Integration in den Entwicklungsprozess
- Integration in den gesamten Evaluierungsprozess
- Unterstützung bei Erstellung nachvollziehbarer Evaluierungsaussagen und Prüfberichte für Evaluierungen

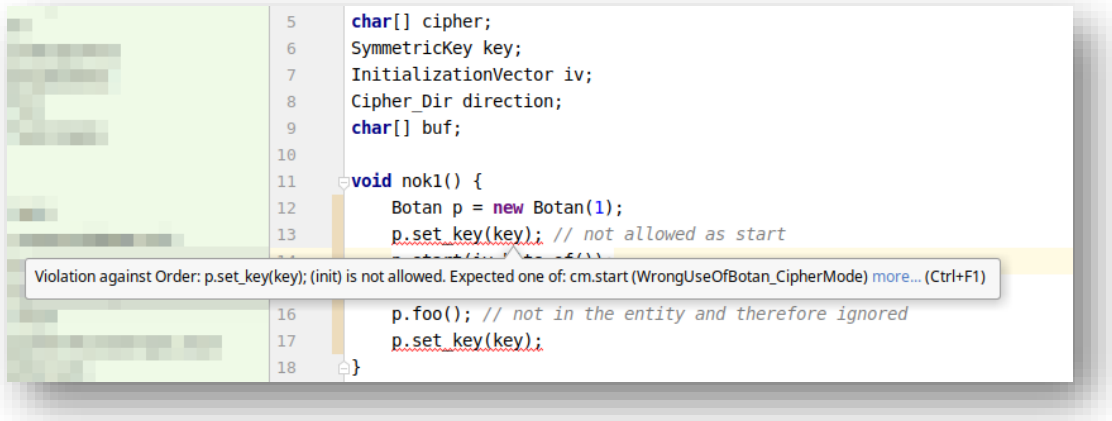


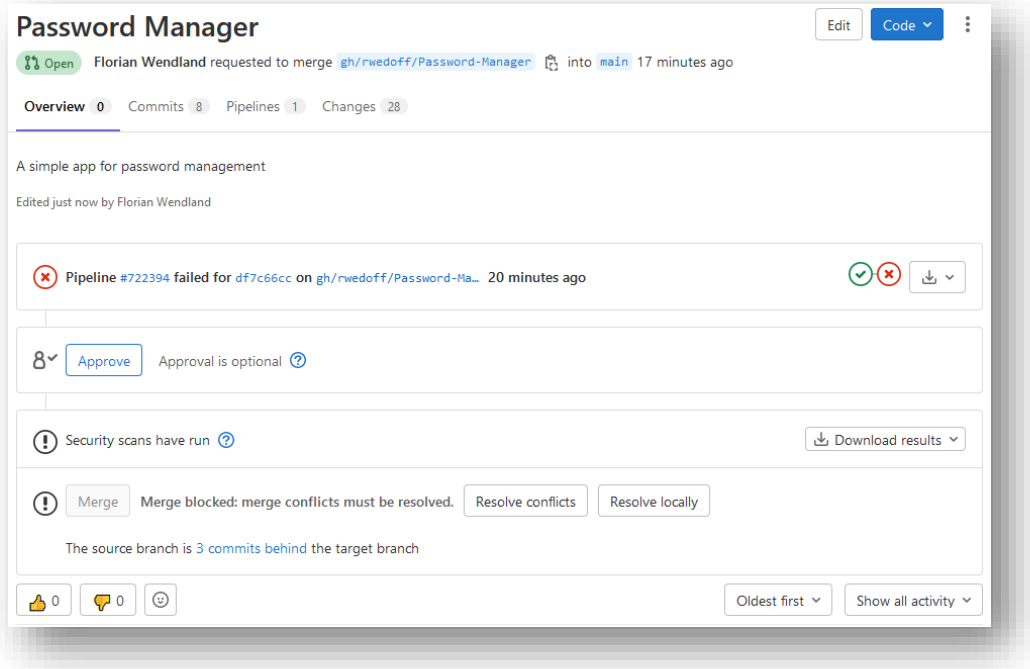
Fraunhofer AISEC

Codyze

Integration in Prozesse

- Integration in IDEs über ein LSP Plugin
  ➔ Support für Eclipse und Visual Studio
- Ausgabe im SARIF-Format
  ➔ Verstöße können in GitHub angezeigt werden
- Start in einem Docker Container
  ➔ Integration in GitHub/Gitlab
- Regeln für Botan2, Bouncycastle JCA und OpenSSL für die TR-2102-1
  ➔ Erste Frameworks sind bereits unterstützt





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

Test it, it's free! **Star it on github!** ★

Fraunhofer AISEC

Demo

Projektkonfiguration durch Konfigurationsdatei

Name	Last commit	Last update
.gitlab	Move to custom GitLab image with Python f...	2 days ago
gradle/wrapper	Example app: Password Manager	2 days ago
mark/gh/rwedoff/password_man...	Activate Codyze	2 days ago
src/main/java/com/github/rwed...	Example app: Password Manager	2 days ago
.gitattributes	Example app: Password Manager	2 days ago
.gitignore	Example app: Password Manager	2 days ago
.gitlab-ci.yml	Update CI pipeline	2 days ago
README.md	Initial commit	1 month ago
build.gradle.kts	Example app: Password Manager	2 days ago
codyze.yaml	Activate Codyze	2 days ago
gradlew	Example app: Password Manager	2 days ago
gradlew.bat	Example app: Password Manager	2 days ago
settings.gradle.kts	Example app: Password Manager	2 days ago

Demo

CI Integration durch Template

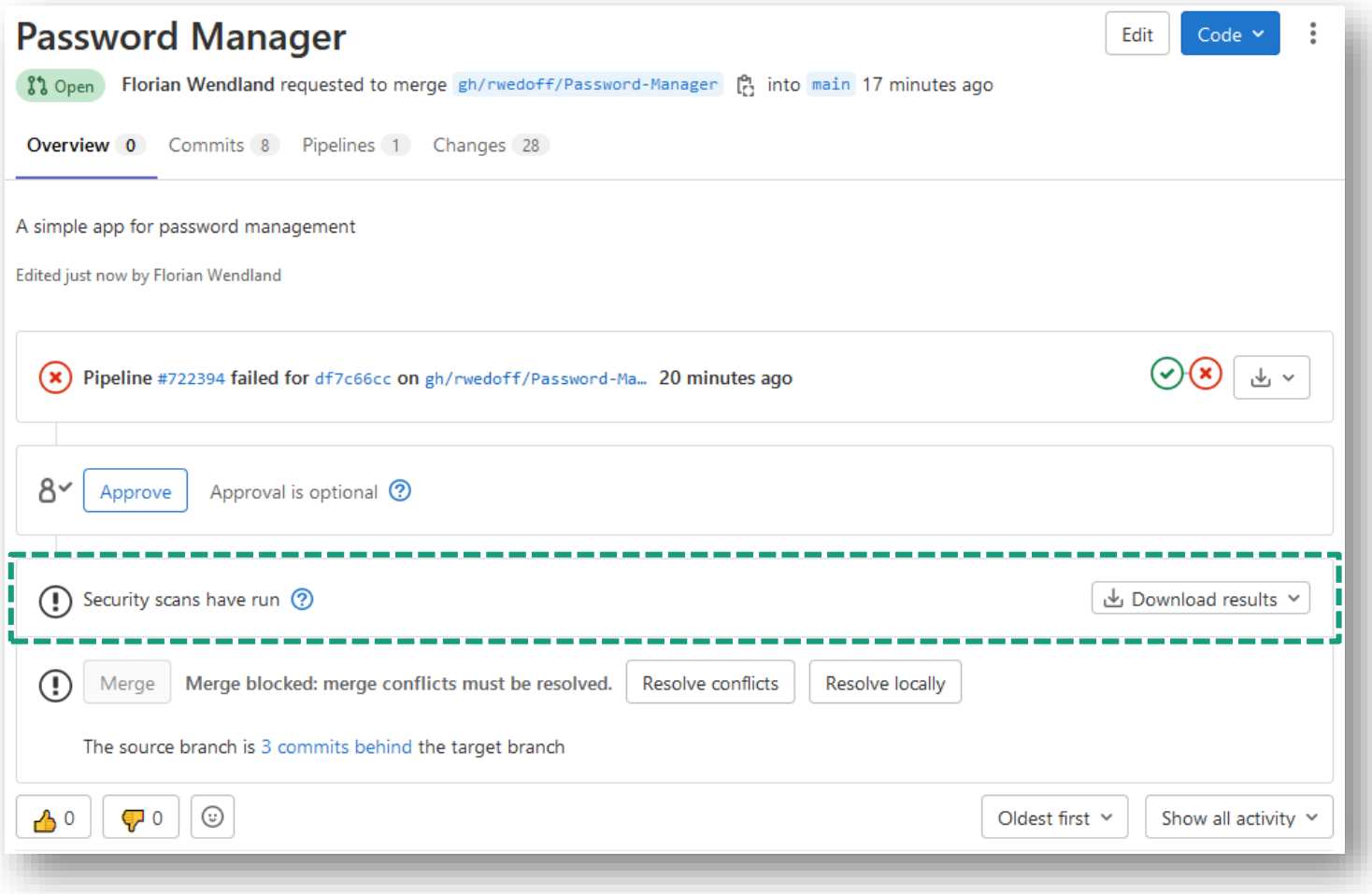
```
default:
  image: eclipse-temurin:11-jdk-alpine
build:
  stage: build
  script:
    - ./gradlew assemble
  artifacts:
    paths:
      - "build/distributions/"
test:
  stage: test
  script:
    - ./gradlew test
  artifacts:
    paths:
      - "build/test-results/"
codyze:
  stage: test
include: '.gitlab/Codyze.gitlab-ci.yml'
```

.gitlab-ci.yml

```
# Codyze template for GitLab CI
codyze:
  stage: test
  image:
    name: codyze/codyze-gitlab:latest
    entrypoint: [""]
  script:
    - codyze --config
  after_script:
    - codyze2gitlab
  artifacts:
    name: codyze
    paths:
      - "codyze.log"
      - "findings.sarif"
      - "gl-codyze-report.json"
    when: always
    reports:
      sast: gl-codyze-report.json
```

.gitlab/Codyze.gitlab-ci.yml

Demo
Checks bei Merge Requests



Fraunhofer AISEC

Demo
CLI & SARIF Ausgabe

```
$ codyze -s src/main/java -m mark/gh/rwedoff/password_manager -o-
{
  "$schema" : "https://json.schemastore.org/sarif-2.1.0.json",
  "version" : "2.1.0",
  "runs" : [ {
    "tool" : {
      "driver" : {
        "name" : "codyze",
        "organization" : "Fraunhofer AISEC",
        "version" : "2.1.0",
        "downloadUri" : "https://github.com/Fraunhofer-AISEC/codyze/releases",
        "informationUri" : "https://www.codyze.io/docs/",
        "rules" : [ ],
        "language" : "en-US",
        "contents" : [ "localizedData", "nonLocalizedData" ],
        "isComprehensive" : false,
        "supportedTaxonomies" : [ ]
      },
      "extensions" : [ ]
    },
    "language" : "en-US",
    "results" : [ {
      "ruleId" : "Wrong_Cipher_Order",
      "ruleIndex" : -1,
      "kind" : "open",
      "level" : "none",
      "message" : {
        "text" : "",
        "id" : "Wrong_Cipher_OrderMessage0",
        "arguments" : [ ]
      },
      "locations" : [ {
        "id" : 0,
        "physicalLocation" : {
          "artifactLocation" : {
            "uri" : "/source/src/main/java/com/github/rwedoff/password_manager/Main.java"
          },
          "region" : {
            "startLine" : 146,
            "startColumn" : 16,
            "endLine" : 146,
            "endColumn" : 70
          }
        }
      } ],
      "rank" : -1.0
    }, {
      "ruleId" : "Wrong_Order_For_SecureRandom",
```


Fraunhofer AISEC

Ausblick

Feedback aus der Pilotierung mit TÜViT

- Interprozedurale Analyse von Konstanten/Variablen ergänzen
- Automatisierte Erstellung von Berichten wünschenswert
- Bessere Beschreibung der Ergebnisse und Ausgaben wäre hilfreich

Ausblick

- Analysen erweitern und verbessern
- Nachvollziehbarkeit der Aussagen erhöhen
- CPG Query API: Abfragen an Code ähnlich zu Prädikatenlogik
  - Allgemeine, wiederverwendbare Abfragen
  - Schneller Überblick über Verstöße
- Neue Anwendungsfälle durch Unterstützung von LLVM-IR
- Idealerweise Bereitstellung von MARK Regeln für verschiedene Bibliotheken über vertrauenswürdige Kanäle

Kontakt

Christian Banse (PL)
christian.banse@aisec.fraunhofer.de

Alexander Küchler
alexander.kuechler@aisec.fraunhofer.de

Fraunhofer AISEC
Service & Application Security
Lichtenbergstr. 11
85748 Garching bei München
https://www.aisec.fraunhofer.de