Towards Generalized Security & Compliance Assessment of Programs

Alexander Küchler, Konrad Weiss, Florian Wendland, Maximilian Kaul, and Christian Banse

Fraunhofer AISEC

Motivation

- Automated analyses are hard to get right and writing them is cumbersome
- Typically, analyses are adapted to different programming languages
- Each programming language has different types, programming paradigms, operations, syntax, behavior of similar concepts, evaluation order, ...

⇒ Abstract from the programming language with minimal information loss

- Standards and regulations have to be met by the developers
- Challenges of automated compliance analysis:
  - Interpreting the high-level text in standard requires expert knowledge
  - Text cannot be interpreted ⇒ Translate to formalized rules
  - Changes in requirements ⇒ Rewrite all rules?

Research-Landscape at Fraunhofer AISEC

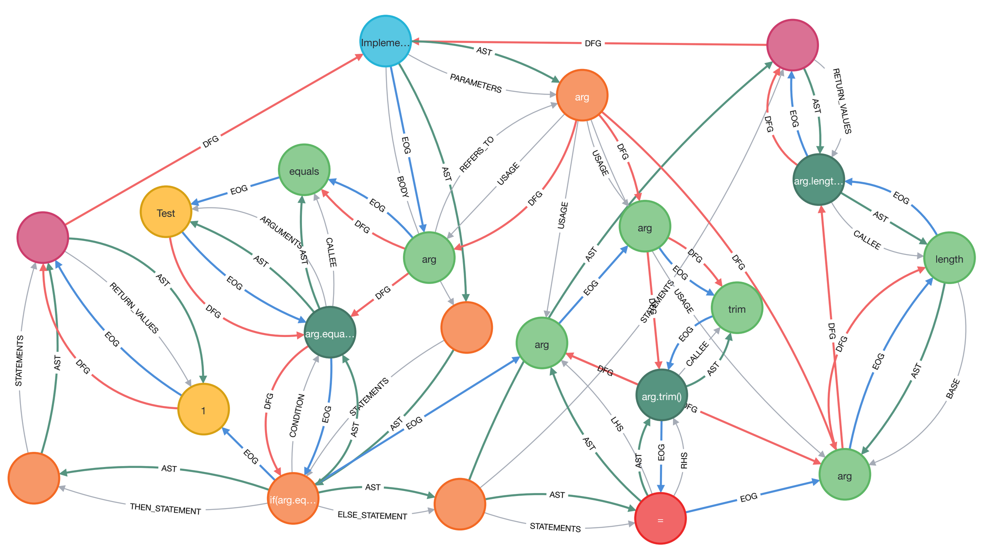
Unified Code Representation

Code Property Graph (CPG)

Generalized representation of code combining

- Abstract Syntax Tree (AST)
- Evaluation Order Graph (EOG)
- Data-Flow Graph (DFG)
- Control & Program Dependence Graph (CDG, PDG)

This is enough information to conduct almost every analysis



Language-Agnostic Representation

- The graph serves as a way to represent and traverse the code
- But how to include the subtle differences between languages?
  - ⇒ *Language traits*: Frequently occurring concepts of programming languages
  - ⇒ Customizable *passes*: deviate from "default" behavior depending on the language
- Extensibility is achieved through a plug-in-like system

Supported Languages

Java, C/C++, Go, Python, TypeScript, LLVM-IR

Built-In Analyses

Dataflow Analysis, Reachability Analysis, Constant Propagation, Intraprocedural Order Evaluation of Statements

Accessing the graph

- Graph DB **neo4j**: Cypher queries and visualization
- **Library**: Integration in other projects
- **Interactive CLI**: Manual exploration
- **Query API**: Built-in analyses and custom queries

Handling incomplete code

Fuzzy parsers: Analyze code with missing dependencies or code fragments

Extensions

- Analysis of cloud applications
- Privacy assessment
- Holistic analysis of quantum programs
- Analysis of Ethereum smart contracts

Abstract Hierarchical Requirements Definition

Codyze

- Defines a set of rules
- Abstracts from concrete implementations
- Parametrizes and runs analyses in the CPG

Correct API Rule Usage

Source of mistakes: Violating conditions when interacting with libraries or APIs, e.g.,

- Input validation
- Typestate: Correct order of operations to bring objects in a specific state
- The choice of arguments might change the security properties

This is specific to a concrete library ⇒ Define library-specific rules.

Use-Case-Specific Standards & Recommendations

- Define what is considered secure today in one area (e.g. cryptography)

⇒ Parametrization of specific API rules (e.g. cipher, mode, keylength, ...)

⇒ Further pre-conditions of parameters (e.g. "key must come from a good RNG")

MARK/CoKo: Language to model such requirements

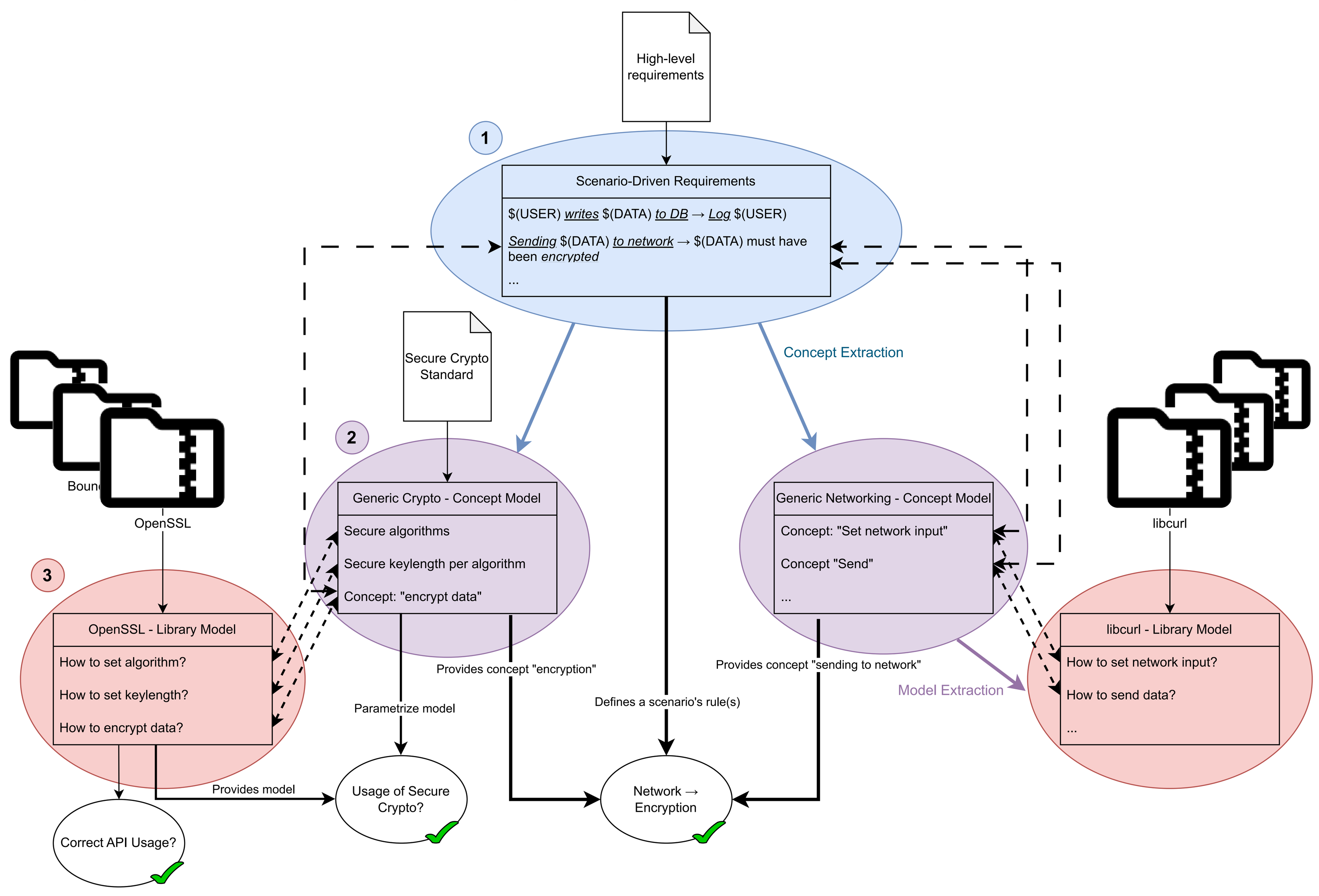
From Specific Libraries to Abstract Concepts

- Other high-level standards use abstract concepts to define scenarios (e.g. "network traffic must be encrypted")
- Can require multiple use-cases or libraries

Conceptualized Analysis of Compliance to Regulations

Goal: Construct a hierarchy of requirements with placeholders:
generic scenarios ① — concepts & requirements ② — implementation ③

- The depth of the hierarchy/tree depends on the use-case to allow maximal flexibility.
- Layer ③ is chosen automatically depending on the code under analysis.
- Layer ② should be customizable e.g. in case of different national standards.



Publications

[1] Christian Banse, Immanuel Kunz, Angelika Schneider, and Konrad Weiss. Cloud Property Graph: Connecting Cloud Security Assessments with Static Code Analysis. In *2021 IEEE 14th International Conference on Cloud Computing*, Chicago, IL, USA, 2021. IEEE.

[2] Maximilian Kaul, Alexander Küchler, and Christian Banse. A Uniform Representation of Classical and Quantum Source Code for Static Code Analysis. In *2023 IEEE International Conference on Quantum Computing and Engineering*, QCE, Bellevue, WA, USA, 2023. IEEE.

[3] Alexander Küchler and Christian Banse. Representing LLVM-IR in a Code Property Graph. In *25th Information Security Conference*, ISC, Bali, Indonesia, 2022. Springer.

[4] Alexander Küchler, Leon Wenning, and Florian Wendland. AbsIntIO: Towards Showing the Absence of Integer Overflows in Binaries using Abstract Interpretation. In *ACM ASIA Conference on Computer and Communications Security*, Asia CCS, Melbourne, VIC, Australia, 2023. ACM.

[5] Konrad Weiss and Christian Banse. A Language-Independent Analysis Platform for Source Code, 2022.

Research Interests & Opportunities

- Enhance analysis capabilities
- Improve/Show the correctness of the representation
- Explore novel use-cases
- Analysis across applications/components
- Modeling standards and libraries
- Identifing concepts and generalizations

More Info

- CPG: https://github.com/Fraunhofer-AISEC/cpg
- Codyze: https://codyze.io
- Fraunhofer AISEC: https://www.aisec.fraunhofer.de

Acknowledgments

This work was partially funded by the EU Horizon 2020 project MEDINA (grant 952633), Bavarian Ministry of Economic Affairs (StMWi), German Federal Ministry of Education and Research (BMBF) project 6G-ANNA (grant 16KISK087), Bundesamt für Sicherheit in der Informationstechnik (BSI).