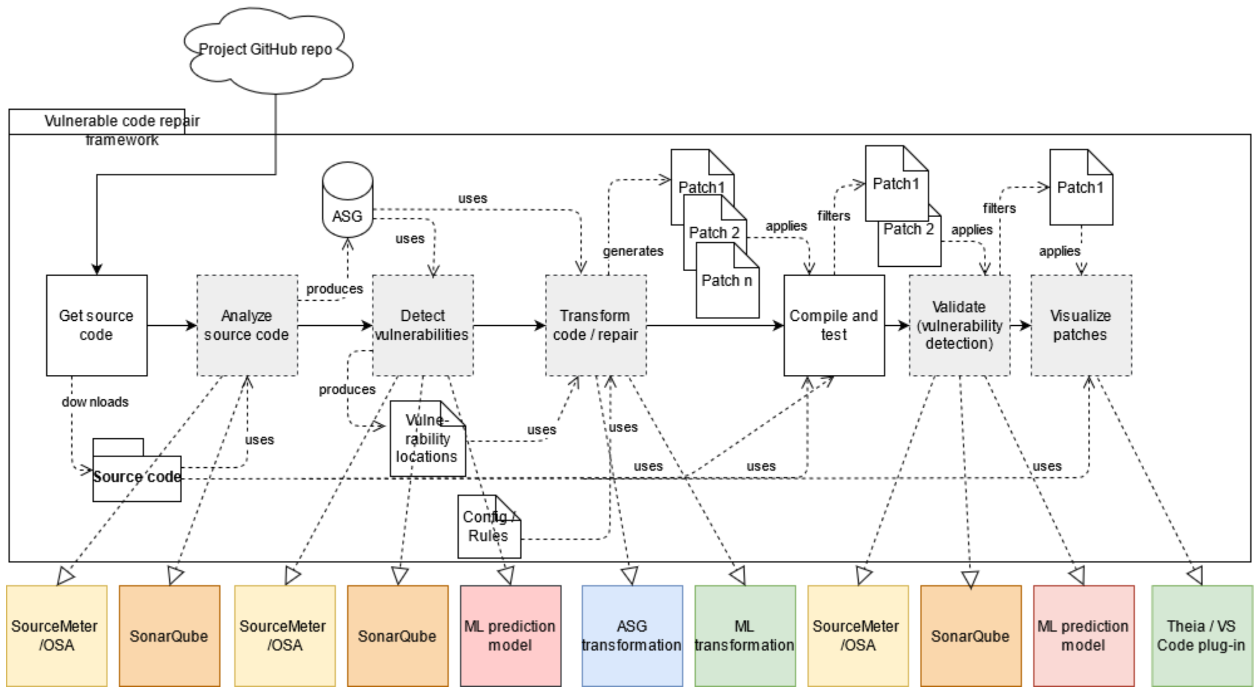
T3.2 Research and Development Concept

Overview



Task 3.2 focuses on automated source code correction in terms of vulnerability removal. As a final output, we plan to build a framework (called **AIFix4SecCode**) and a tool, which orchestrates the whole process of the localization and correction of security vulnerabilities, then the visualization of the generated fixes. The architecture of AIFix4SecCode is modular and supports a plug-in mechanism. The modules marked with gray and dashed lines in the above figure are exchangeable, define only the appropriate interfaces for input and output and can be freely substituted with arbitrary implementation. For example, to get the vulnerability locations in the source code, one can apply a static analysis based vulnerability detection but just as easily apply a ML prediction model to find vulnerable code parts.
The tool would be explicitly run by the developer and then present the corrections (maybe more generated patch candidates) as diffs so that the developer can clearly see what is intended be modified and make the final decision to accept, modify or reject the changes proposed by the tool.

Module description

Get source code. This module is responsible for acquiring the source code from a remote repository and store it into a local folder (i.e., a git clone, but other sources might be considered).

Analyze source code. The module that takes the source code as input and produces artefacts, like representation of the source code (ASG, embedding, etc.) to be used later in the detection and repair step. As a first step, we implement this module with our code analysis tool SourceMeter/Open Static Analyzer[1] that integrates many checkers, like SpotBugs, SonarQube, PMD that are able to detect vulnerabilities.

Detect vulnerabilities. The module that takes the source code as input and produces the list of source code positions where vulnerabilities are detected. As a first step, we implement this module with our code analysis tool SourceMeter/Open Static Analyzer[2] that integrates many checkers, like SpotBugs, SonarQube, PMD that are able to detect vulnerabilities. Later on, we add a prediction model implementation for detecting vulnerabilities as well.

Transform code / repair. This is the core module of the framework that performs the actual code repair task. It uses the source code of the project and the intermediate representations produced in the previous step (e.g. an ASG). This module can load and use some configuration from external files as well (i.e., transformation rules). We implement an Abstract Semantic Graph (ASG)-based transformator that performs code changes at the ASG level and generates source code (Java) based on this transformed ASG. This ASG transformation can be pattern-based or learned from the data collected in T6.1, in which case we would generate the ASG representation of both states of the code (vulnerable and fixed) and define (learn) the transformation steps necessary to correct the security issue. The implementation of this module might mix strategies as well, for example, for certain types of vulnerabilities a pattern-based ASG transformation is applied, while for other types of issues, for which plenty of training data exists, an ML based transformation method can be applied. The output of this module is a set of diff files containing the automated code corrections proposed by the framework.

Compile and test. After the code repair candidates are generated, we need to validate that the proposed code changes keep the system in a syntactically valid state with all the unit tests passing. This module will perform these sanity checks. The supported build systems can be extended as well, by implementing the proper interfaces of this module. We plan to add support for maven first, but extend it later on. After the module is executed, some patches might be thrown away that produce failing tests or break the syntax of the code.

Validate. After filtering out clearly invalid patches, we need to validate that the originally detected vulnerability really disappears after applying the generated patch. We achieve this by rerunning the same analysis as in the step of vulnerability detection and confirm that the targeted vulnerability is not found any more in the source code version after applying the fix patch. The output of this module is the filtered list of patches that indeed remove the vulnerability in question.

[1] https://github.com/sed-inf-u-szeged/OpenStaticAnalyzer
[2] https://github.com/sed-inf-u-szeged/OpenStaticAnalyzer

Visualize patches. This module implements the last step of the repair process, the visualization of the code changes. The input of this module is the final set of patches that the module visualizes. The user interface that employs such a tool would be a parallel development process. Currently we plan to create an Eclipse Theia[3] plugin to achieve this goal.

Framework definition

JAVA application

Interfaces

SourceCodeCollector

Clones git repository
Input: URL
Output: path to folder

VulnerabilityDetector

Concrete implementation is a wrapper for SourceMeter/Open Static Analyzer, SonarQube, ML prediction model, etc.
Input: path to folder
Output: vulnerability locations (file containing filename, line number, vuln reference?)

CodeAnalyzer

Creates the ASG/Control Flow, embedding
Input: path to folder
Output: Code model object in memory

VulnerabilityRepairer

Input: vulnerability locations, Code model object
Output: patched files and patch list (list paths to patched files, can contain multiple variants)

PatchCompiler

Applies patches based on patch list
Checks whether compilation was successful for generated patches, filter non-compiling failed test cases from the variants
Input: source code, patch_lists

[3] https://theia-ide.org

Output: filtered patch list (successfully compiled patches)

PatchValidator

Runs Detector for patch candidates and compares what vulnerabilities are corrected successfully
Input: original vulnerability locations file, source code, filtered patch list
Output: validated patch list

PatchVisualizer

TBD
https://github.com/moshfeu/vscode-diff-merge

VulnRepairDriver

```
Implements pipeline, instruments tools
Read config file
    Config contains:    repo location
                        Concrete types for modules (e.g. SourceMeter as Detector)
If repo location is URL:
    Run Collector
Run CodeAnalyzer
Run VulnerabilityDetector
Run VulnerabilityRepairer
Loop over patch list
    Run PatchCompiler
Run PatchValidator
Run PatchVisualizer
```