UNIVERSITÄT BONN

Secure Software Engineering

Exercise Sheet 7, Winterterm 25/26

Discussion Week: 05.01.26 to 09.01.26

We highly encourage you to do the assignments yourselves, as all the exercises are relevant for the exam. It is also recommended to visit the tutorial, as there will be a discussion about the exercises, not just the solutions. For the *Vulnerability of the Day* exercises, we recommend examining the code snippets, if any, while conducting your research. Keep in mind that the entirety of the lecture is relevant to the exam. If a topic or some details are not specifically talked about or mentioned in the exercises, that does **not** mean that it will not be part of the exam!

If you have any questions, please do not hesitate to ask your tutor Aura, Kati or Lukas. (Hint: the mails are embedded in the names.)

Good luck and have fun! :)

Ex. 1 - Code Scanning Fundamentals

Answer the following sub-tasks:

a) Name the three types of graphs that are combined into a Code Property Graph (CPG). For each, briefly state what it captures that the previous one cannot.

b) What is the difference between SAST and DAST? Name one advantage and one disadvantage of each.

c) Explain what fuzzing is and what the two main components of a fuzzer are.

Ex. 2 - Writing Custom SAST Rules

In this exercise, you will write custom static analysis rules for a proprietary database driver. Download the repository from anonymous.4open.science (click "Download repository" in the top right corner).

Scenario: Imagine a company has developed their own database driver called `Database` (similar to JDBC). Since it's proprietary, no existing Semgrep or Joern rules exist to detect SQL injection vulnerabilities in code using this driver. Your task is to write these rules.

Note: The `Database` implementation is just a wrapper around JDBC for demonstration purposes. You can ignore its internals. Focus on the application code in Main and Examples that *uses* the driver.

Tasks:

a) **Explore the Code Property Graph:** Use Joern to generate and explore the CPG of the vulnerable application code.

- Import the project into Joern
- Dump the CPG and explore its structure
- Identify the sources (user input) and sinks (database queries) in the graph
- Document your observations about how data flows from input to query execution

b) **Write Detection Rules:** Create rules to detect SQL injection vulnerabilities in code using the `Database` driver. Use the provided templates:

- Write a *Semgrep rule* in `rules/sql-injection.yaml` that detects string concatenation in SQL queries passed to `Database`
- Write a *Joern query* in `queries/sql-injection.sc` that uses taint analysis to find paths from user input to unsafe query execution
- Test your rules against the provided vulnerable and safe code examples

Ex. 3 - Dynamic Analysis with Burp Suite

In this exercise, you will get hands-on experience with Burp Suite, a popular DAST tool. Follow the official getting started tutorial and connect what you learn to the lecture concepts.

Setup:

1. Download and install Burp Suite Community Edition
2. Complete the getting started tutorial: portswigger.net/burp/documentation/desktop/getting-started

Tasks:

a) Complete the *Intercepting HTTP traffic* and *Modifying HTTP requests* sections. In the lecture, we discussed that DAST tools use *fuzzing* (mutated inputs) and *attack heuristics*. How does Burp Suite's Repeater tool allow you to manually apply these techniques?

b) Complete the *Scanning a website* section. Burp Scanner automates the two DAST components from the lecture: *input generation* and *bug detection*. Explain where in the tutorial which was used.

c) Based on the tutorial and lecture, name two types of vulnerabilities that Burp Suite's scanner looks for and explain what *heuristics* (attack patterns) it uses to detect them.