Understanding and Using Static Code Analysis Tools

The goal of this assignment is to familiarize you with using open source static code analysis tools to find bugs and vulnerabilities in source code and to recognize that not all tools are created equal. Each tool uses different analysis techniques and is programmed with different rules for detection of problems. Also, the settings, and in some cases the plugins, used in a tool will have an effect on what types of problems it can identify.

This will be an individual assignment (no teams).

Part 1 (85 Points Max):
For the first part of this assignment, you will analyze a simple web server written in Java that is posted on Blackboard along with this document. The analysis will be performed with at least two tools, one of which must be SpotBugs (located at https://spotbugs.github.io), which is the continuation of FindBugs, with the Find Security Bugs Plugin (located at http://find-sec-bugs.github.io/). The second tool can be a tool of your choosing (e.g. PMD v5.5.2, JLint 3.0, SonarLint 2.1, Infer, etc.). Some tools, such as SpotBugs, analyze the binary (Java bytecode) while others analyze source directly (the documentation for the tools should provide direction as to whether the tool analyzes source or binary). As a result, you will be required to build the source for the simple web server in order to analyze it with SpotBugs.

The following web site is a good starting point for locating static analysis tools for various languages: https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html.

Prior to using a tool to analyze the simple web server, manually review the code and see if you can identify any problems. Document any problems you find. Explain where in the code the problem lies (file and line number) and why you think it's a problem. Don't worry if you don't catch very many. Consider this a baseline assessment of your existing code review skills and ability to spot security problems. As you look at the code, think about ways an attacker can exploit weaknesses in the web server.

Run the analysis across the entire code base (in this case, a single Java class). Each tool typically has a variety of ways it can be invoked (command line, from scripts, integrated into build systems such as Ant or Maven, from integrated development environments such as Eclipse, etc.). The method of invocation you use is up to you, but you should be able to understand the requirements for the tool, the types of files on which it operates, and what it produces. **Make sure you turn on scanning for security-related vulnerabilities and/or enable security rule sets as appropriate in the chosen tool.** Document any tool findings and indicate which ones you found in your manual analysis.

Experiment with changing the sensitivity levels within each of the tools (this setting might be called one of many things and has to do with how aggressive the tool is in performing an analysis). Notice the increase/decrease in the number of findings reported. Also, experiment with turning on/off various rule sets in each tool. Document your observations in the report. NOTE: Based on the small code sample being used, you may or may not notice a difference in the findings.

The graduate teaching assistant should be available to provide assistance as needed in setting up your environment and building the simple web server if you are struggling to get it working.

Part 2 (15 Points Max):
For the second part of this assignment, pick some code you've written in the past (possibly homework for another class). It can be in any language (as long as you can find a capable static code analysis tool for it). C/C++ or Java would be preferable. Do a manual code analysis and document any findings. Then run an analysis tool on it to see what kind of findings you get and try fixing a few of the bugs. The goal here is to make you aware of vulnerability escapes in your own prior work which may lead to malicious exploitation. It will help you uncover mistakes so that you can adjust programming practices to avoid them in the future.

Submission:
Submit a zip file with the following contents:

- You will submit a report in PDF format that documents the following:
  - Results of your manual review of the simple web server and your own code (conducted prior to running the tools).
  - Your tool choices and versions.
  - Your approach to invoking the tools. Include screen shots and steps.
  - Your understanding of how the tools work (compare and contrast each tool) as well as differences in types of flaws reported. This will come from reviews of the documentation for each tool.
- Include attachments of tool output reports, fixes (for your own code), preferably in HTML, PDF, Excel, or ASCII text (XML as a last resort).

Grading Criteria:

Part 1 (85 points max)

- Manual Analysis (5 points total)
  - Included 2 points
  - Legitimate Findings 3 points
- Tool Choices/Versions 5 points
- Tool Invocation Process (20 points total)
  - Steps 10 points
    - Should include information about tools settings and plugins used.
  - Screenshots 10 points
- Comparison/Contrast Tools 20 points
  - Does the tool analyze source or binary as input?
  - Which category of tools is it?
    - Type checking
    - Style checking
    - Program understanding
    - Program verification

        - Property checking
        - Bug finding
        - Security review
    - Show an example (if one exists) of a finding that is reported by one tool and not others.
    - Show an example (if one exists) of a finding reported by multiple tools.
    - For the known flaw in the code used, document which tools reported it (true negative) and which tools did not (false positive).
- Results 40 points
    - Raw results provided (in separate file, directly output from tool).
    - Security rules enabled (along with required security plugins).
    - Turned on most aggressive mode in tool for finding defects.

Part 2 (15 points max)

- Code Provided 2 points
- Manual Analysis (5 points total)
    - Included 2 points
    - Legitimate Findings 3 points
- Results 5 points
    - If none found (code perfect?), at least state so.
- Fixes 3 points
    - If no fixes to make, at least state so.
    - If potentially lots of flaws, full points if at least 2-3 are fixed.