Tool	Languages	IDE integration	Analyses
Flawfinder	C, C++	vim, emacs	lexical scan
Cppcheck	C, C++	Visual Studio	path-insensitive dataflow
Spotbugs/ FindSecBugs	Java	Eclipse	structural data/control flow
SonarQube	multi-language	CI/CD Tools	only simple analyses for free

PVS-Studio	C,C++,Java	Visual Studio, Eclipse, ..	Data/control flow, symbolic execution

Static Analysis Tools

- Examples (open source)
- Properties
  - Open source/commercial
  - Techniques uses/detection capabilities
    - Security vulnerability coverage
    - FP/FN benchmarks
  - Programming Languages supported
  - IDEs Integration
- Techniques
  - Lexical Scanning
  - Structural checking
  - Data/Control Flow Analysis
  - Temporal Logic Checking
    - Model Checkers
    - Theorem Provers
  - Symbolic Execution
  - Concolic Execution
- Evaluation Metrics
  - Precision in modeling software behavior
  - Depth/Scope: consider only relationships between lines of code in the same
    - line
    - function
    - module
    - application
  - Execution time/scalability
    - ms
    - s
    - minutes
    - hours
    - unfeasible
- Accuracy measurement
  - True Positive Rate=True Positives/ (True Positives + False Negatives=Total Positives) -> ability to find vulnerability (maximize)
  - False Positive Rate=False Positives/(False Positives+True Negatives=Total Negatives) -> low FPR means ability to avoid reporting false vulnerabilities (minimize)
  - OWASP Benchmark Project: syntetic programs
  - NIST Software Assurance Reference Dataset: real vulnerable programs
  - Web Application Vulnerability Scanners Evaluation Project: dynamic analysis evaluation
- Workflow
  1. Decide tool to use
  2. Decide which vulnerabilities you are looking for and corresponding configuration
  3. Run tool
  4. Get raw result: tool report
  5. Perform manual code review to check all the reported vulnerabilities and divide them in TP or FP
  6. Report TPs in bug tracking systems with problem explanation with reference to the security policy and risk estimate and annotate reason for FP on the code to avoid checking them again in the future
  7. Attacker develops exploit based on vulnerability/ Developer fixes code
  8. Run the tool again to check if the vulnerability was fixed correctly