Analysis Tools

CIS-3720, Secure Programming

Peter Chapin

Classification of Tools

- There are several approaches to analysis
  - Language extensions
    - Use a specialized dialect of the language that includes additional information
    - Modified compiler (or external tool) reads that additional information
    - PROs: Potentially very powerful
    - CONs: Requires special compiler (might fall behind), code no longer compatible with standard tools (compilers, IDEs, other analysis tools, etc.)
  - Standalone tool
    - Analyzes the standard language (perhaps with “annotations” in comments)
    - Does not require specialized compiler or tools
    - May have limitations as to what can be done
    - **By far, more common approach**

Classification of Tools

- When is the analysis done?
  - Statically (before the program runs)
    - PROs: Analyzes, in effect, every possible execution; can be made part of the build and continuous integration process; entails no runtime penalty.
    - CONs: In general, not precise (false positives + false negatives); increases development time (some analyses are very time consuming to execute)
  - Dynamically (while the program is running)
    - PROs: Perfectly precise **for the executions tested**; can “see” effects due to the program’s environment
    - CONs: Runtime overhead; analysis done as part of testing (rather than construction); relies on good test suite

Examples

- Jif: Java + Information Flow
  - Extended version of Java with additions to express secure information flow
    - Allows user to statically check flow of confidential and tainted data in a program
    - Program is totally incompatible with standard Java due to the additional syntax
- SpotBugs
  - Static analysis of Java class files for various bug types
    - Works with standard Java (really any language that targets the JVM)
    - Like all static analyses it is imprecise
- Splint
  - Static analysis of C programs for various bug types and security vulnerabilities
    - Works with Standard C99, not C++