Automated testing for software vulnerabilities

Adapting reliability testing techniques to the study of security properties
Security issues in software

- Security flaws in software can be introduced at many levels
  - Incorrect capture of security requirements/ lack of a security model (specification)
  - Incorrect choice of implementation strategy for a given set of requirements (design)
- Improper design implementation, either by incorrect logic, or by failing to take into account important details from the environment
- **Software defects, errors, no enforcement of input format restrictions, unsafe manipulation of inputs and/or internal state**
Reliability testing techniques

- The latter category of vulnerabilities (defects and unsafe operations) are also associated with reliability issues, and may affect the robustness of software.
- Several techniques exist to facilitate the testing of software for instability and defects, to assure quality and reliability.
Software analysis terminology

• **Black-box analysis:**
  – Explores properties of the software behavior and configuration exclusively, without directly using information from source code or binaries.

• **White-box analysis:**
  – Explores properties of the code, in addition to program behavior. Typically implies access to source code, but also performed on binaries.

• **Static analysis:**
  – Studies properties of the software and its configuration from the source code, binaries, API, configuration files, etc. The software is not executed.

• **Dynamic analysis:**
  – The software is executed in a variety of environment and input scenarios, and the behavior is studied for conformance with specified requirements.
Automating analysis/testing

- Automated static code analysis involves scanning the source code and its binary to search for known vulnerability patterns, such as use of unsafe buffer operations that can lead to buffer overflow, implicit or explicit cast that may trigger rounding and modification of program logic, etc.

- Automated dynamic tools tend to exercise the program on a large number of inputs/environmental settings to identify possible errors and quality issues
Some methods to improve software quality

- Debuggers
- Configuration management
- Memory leak detection
- Performance profiling
- Load testing
- Analysis of structural metrics
- Test case generation
- Code coverage analysis
Fault injection characteristics

• May be white-box or black-box testing
• Dynamic testing technique involves program execution
• Can detect defects both of software and its configuration
• Originally developed for reliability, some adaptations necessary for effective use in inspection of security properties
Fault injection for security: How it works

- Identify security properties and express this in terms of system invariants (e.g., protected file remains unchanged)
- Injects faults
  - Directly manipulate the internal variables, or provide a variety of (potentially erroneous) inputs
- Force program to assume anomalous states
- Measure effects during execution. Was the particular security property preserved?
Differences from reliability testing

• Again, fault injection evolved first with the purpose of quality/reliability testing
• In the case of security analysis, differences appear
  – Reliability tools tend to generate inputs/ perturbations at random, or following a known distribution, while security attacks are designed and do not need to follow statistical rules.
    • To use such tools most effectively it pays to have some insight about the types of inputs that could lead to vulnerabilities
  – Violation of security properties may not lead to any obvious departure from program behavior and may go undetected by not leading to instability. Explicit tests for each security property must be instrumented manually by an expert.
AVA algorithm

- Adaptive Vulnerability Analysis
  - Executes the software
  - Injects faults during program execution
  - Determines violation of security properties
Steps of algorithm

- Annotate by hand all the locations of the code where to test for preservation of security properties, or choose all locations (instructions).
- For each location \( l \), set counter \( c_l \) to 0.
- Randomly select an input from one of several possible distributions.
- Alter the value of one of the internal variables, again following particular distributions.
- Test if a violation of the security predicate occurs (some assertion about security invariants, e.g., protected file unmodified). If so, increment the counter at that location.
- After \( n \) executions, count accumulates how many runs lead to security violation. \( c_l / n \) is an indicator (security assessment), associated to the location, the internal variable sampled and the security property investigated.
Fault injection engine

- To implement the AVA algorithm, a fault injection system (FIST- Fault Injection Security Tool) was developed

- Instruments (annotates) every instruction--so a particular security fault may be detected anywhere in the code

- Program states are perturbed singly in each test run
  - However, some security flaws may require multiple perturbations before a violation of the security property occurs
Types of faults

- Perturbations of all primitive data types
  - Boolean flips, random character substitutions, string mangling (truncate, concatenate random string, replace strings with randomly generated ones or with ones read from a file), …
  - Stack smashing buffer overflow functions
  - Composition of faults
Buffer overflow fault

- The buffer overflow perturbation overwrites the return address of the stack frame in which the buffer is allocated with the address of the buffer itself.
- Opcodes for machine instructions are written into the buffer being perturbed.
- Checks if the software constrains the input values written to the buffer.
Security policy

• In order to use FIST/AVA effectively, it is necessary to have a model of security for the program
• The policy must be translated into a number of pre-defined assertion types
  – Arbitrary integer-valued C expression
  – File content modification/ file access time
    • The latter implements checks for the buffer overflow function, as the opcodes introduced in the buffer try to modify a specially named file. The assertion “file unmodified” is then sufficient to check for buffer overflow vulnerabilities