Security Code Testing Methods

AUBURN UNIVERSITY

Holistic Approach To Security Testing

AUBURN

- Secure software applications include all types of testing: static analysis, dynamic analysis, manual code review, penetration testing, and fuzz testing
- Other mitigating practices in addition to software testing:
  - Mandatory software security training
  - Security design reviews
  - Threat modeling,
  - Measurable criteria
  - Extending practices into including servicing and support.
- Consequences of not performing these practices due to time and resource constraints are significant:
  - Cost to fix increase the further downstream in development process or worse in production
  - Damage to customers
  - Damage to brand name of the company

Types of Security Testing

AUBURN

- **White Box**
  - Testing with internal knowledge on structure of software
  - Language, code structure, design documents, dependencies, and configuration files
- **Gray Box**
  - Partial knowledge of source code to design the test cases. Focused on integration layer or specific layers of application (UI, Persistence/DB, API)
  - Uses black-box testing techniques on the integration layers.
- **Black Box**
  - Testing the software externally with no knowledge of internals
  - Examples are:
    - HTTP for websites
    - Network TCP ports for Servers
    - Externalized APIs

Testing Methods

AUBURN

- **Manual Code Review -** typically line-by-line inspection of source code based on knowledge of coding standards, common vulnerabilities, and software design
  - Can be in conjunction with SAST findings
- **Static Application Security Testing (SAST)** - testing software source code while not being executed.
  - Code Scanning Tools: SonarQube, HP Fortify
- **Dynamic Application Security Testing (DAST)** - testing software while it is in running environment executing source/machine code in a real or virtual process.
  - Vulnerability Scanning Tools: Fortify, Burp Suite
  - Penetration Scanning Tools: Wireshark, Tenable, Burp Suite
- **Fuzz Testing -** Black-box software testing that provides invalid, unexpected or random data to the inputs of a software system.
  - Uses malformed/semi-malformed data injection in an automated manner.
  - Results are then monitored for exception returns such as crashes, failing built-in code assertions, and potential memory leaks.

Key Aspects of Fuzz Testing

AUBURN

- Black box method of testing - does not know source code
- Attack simulation in which unexpected data (invalid, unexpected, random) is fed to the system through an open interface, and the behavior of the system is then monitored.
  - crashes
  - failing built-in code assertions
  - potential memory leaks
- Issues found by fuzzing tools are critical and exploitable, however theyonly find bugs that can be accessed through an open interface
- Automated or semi-automated (manually executed)
- Popular tools:
  - Codenomicon (commercial)
  - Peach Fuzzing Tool (open source)

Smart & Dumb Fuzz Testing

AUBURN

- Smart
  - Although all of the issues found by fuzzing tools are critical and exploitable, unlike static analysis tools, fuzzing can only find bugs that can be accessed through an open interface.
- Dumb
  - the fuzzer systematically pushes data to the program without waiting for proper responses. This method is closely tied to denial-of-service attacks. This method requires no knowledge of the target and uses existing tools. However, more crash analysis is required, and there is more duplication of findings than with “smart” fuzzing.

Comparing Testing Methods

AUBURN

Method		Advantage	Disadvantage
Manual Code Reviews (SAST)	White Box	• Takes architecture and functionality into consideration • Requires no technology tools • Promotes teamwork & learning • Can be performed early in the development cycle	• Time consuming & costly • Slow compared with automated tools • Effectiveness based on people skills and experience
Static Analysis (SAST)	White Box	• Access to Source Code • Cover entire code base • Identify exact location • Fast if automated • Quick turn around for fixes • Tools provide mitigation	• Access to source code • False positives/negatives • Weak on operation/runtime issues • Manual trigger slow • False sense of security once automated • Only as good as configured rules
Dynamic Analysis (DAST)	Gray Box Black Box	• No requirement for source code • Fast if automated • No need to understand software code • Runtime environment vulnerabilities • Can validate static code findings • Can be conducted on various technology stacks. Not language specific	• Requires running environment • Writing test cases could take time • False sense of security once automated • Only as good as configured or written tests • Hard to trace vulnerabilities to code location • Requires trained people to run and analyze • Domain knowledge of application required
Fuzz Testing (DAST)	Black Box	• Test design is simple • Random approach find bugs that would often be missed • Identified issues are typically severe • Good for testing closed systems.	• Finds simple issues • Poor code coverage

AUBURN UNIVERSITY