Toshi’s Approach to Runtime Analysis

Black Box Scanning Tool

+ 

White Box Testing Tool
Toshi’s Black Box Scanning Tool

- **Same approach as:**
  - Cenzic
  - SPI Dynamics
  - Watchfire

- **Toshi’s tool is unique because:**
  - Built on Microsoft Visual Studio 2005 platform
  - Reuses Web application testing capabilities
  - Builds on existing test scripts
  (not useful today; we didn’t give him any scripts)
Black Box Scanning

1) Traversing the application
   • Manual
   • Automated

2) Testing the application
   o Signature analysis
   o Behavioral analysis
Traversing: Manual Crawl

- Manually map the application’s interface

**Advantages**
- Can often achieve higher coverage

**Disadvantages**
- Time consuming
Enter starting URL and map the interface automatically

Advantages
- Easy to use
- Sometimes comprehensive

Disadvantages
- Cannot crawl complex web applications
- Make take a long time, looping redundant pages
Black Box Scanning

1) Traversing the application
   o Manual
   o Automated

2) **Testing the application**
   - Signature analysis
   - Behavioral analysis
Testing: Signature Analysis

- Search for specific strings in the HTTP response

- Example: SQL injection
  - “SQLException”
  - “OLE DB Provider”
Testing: Behavioral Analysis

- Identify behavior indicative of a vulnerability

Example: Blind SQL Injection
1. Inject original clause: id=3
2. Inject true clause: id=3 AND 1=1
3. Inject false clause: id=3 AND 1=0
4. If 
   
   (original==true && true != false)

then report SQL injection
Advantages Black Box Scanning

- **Advantages**
  - If you have a running application, you can test it
  - Bugs are easy to verify (reproduce)
Disadvantages: Low Coverage

- You can’t test what you can’t reach
Disadvantage: Missing Oracles

- Some vulnerabilities not visible from Web

I hope they're not logging my CC# into plaintext log file

Client → Application

Credit card # → Application

HTTP Response → Application

“Your order will be processed in 2 days”

CC # → Log File
Toshi’s Special Sauce: White Box Testing Tool

- Insert monitors around security-relevant APIs
  - Sources of input
    - Web: ServletRequest.getParameter(String)
  - Sinks
    - Database: SQLStatement.executeQuery(String)
    - Process: Runtime.exec(String)
    - File: Log.log(String)
- Look for potential problems
Combats Black Box Limitations

- **Coverage**
  - Percentage of security-relevant APIs exercised

- **Code-level details**
  - File name, line number and API details for bugs

- **Improved oracles**
  - Vulnerabilities not evidenced on Web
Black Box Scan + White Box Testing Tool

- Web Scanners
- HTTP
- Application Server
- Web Application
- Database
- File System
- Other Apps
- Verify Results
- Watch Result
How To Inject Monitors

- Monitor code written as aspects

- Use aspect-oriented technology
  - AspectJ (Java)
  - AspectDNG (.NET)

- Works on bytecode
  - Java class files & .NET MSIL

  (no source code required)
Bytecode Injection: Process

Original .class → AspectJ → New Code and Location → New .class
Bytecode Injection: Result

List getStuff(String id) {
    List list = new ArrayList();
    try {
        String sql = "select stuff from mytable where id='" + id + "';
        JDBCstmt.executeQuery(sql);
    } catch (Exception ex) {
        log.log(ex);
    }
    return list;
}

List getStuff(String id) {
    List list = new ArrayList();
    try {
        String sql = "select stuff from mytable where id='" + id + "';
        MyLibrary.doCheck(sql);
        JDBCstmt.executeQuery(sql);
    } catch (Exception ex) {
        log.log(ex);
    }
    return list;
}
Summary

- Black box scanner
  - Smart fuzzer (uses specific attack strings)
  - Oracles with signatures and behavioral analysis

- White box testing tool
  - Inject monitors
  - Provide coverage, code details, enhanced oracle