Comparing the Effectiveness of Penetration Testing and Static Code Analysis

Detection of SQL Injection Vulnerabilities in Web Services

Nuno Antunes, Marco Vieira

nmsa@dei.uc.pt, mvieira@dei.uc.pt

University of Coimbra – Portugal
Web Services

- Web services are becoming a strategic component in a wide range of organizations
- Components that can be remotely invoked
  - Well defined interface
- Web services are extremely exposed to attacks
  - Any existing vulnerability will most probably be uncovered/exploited
- Both providers and consumers need to assess services’ security
Web Services Environment

Diagram showing the flow of communication between a consumer, registry (UDDI), and providers (0, 1, 2). The consumer interacts with the registry and providers through WSDL and SOAP requests and responses.
SQL Injection vulnerabilities...

```java
public String auth(String login, String pass) throws SQLException {
    String sql = "SELECT * FROM users WHERE " +
    "username='" + login + "' AND " +
    "password='" + pass + "'";

    // Example of SQL injection vulnerability
    sql = "SELECT * FROM users WHERE username='" OR 1=1 --' AND password='"; // Malicious query
}

public void delete(String str) throws SQLException {
    String sql = "DELETE FROM table WHERE id='" + str + "'";

    // Example of SQL injection vulnerability
    sql = "DELETE FROM table WHERE id='" OR '' = ''; // Malicious query
}
```
Developers must...

- Apply best coding practices

- Perform code analysis
  - Manual code analyses (reviews, inspections)
  - Automated static code analysis

- Perform tests
  - Manual penetration testing
  - Automated penetration testing (vulnerability scanners)
Penetration testing

- Widely used by developers

- Consists in stressing the application from the point of view of an attacker
  - “black-box” approach
  - Uses specific malicious inputs
    - e.g., for SQL Injection: ‘ or 1=1

- Can be performed manually or automatically
  - Many tools available
    - Including commercial and open-source

- Does not require access to the code
Static code analysis

- “white-box” approach
- Consists in analyzing the source code of the application, without execution it
- Looks for potential vulnerabilities
  - Among other types of software defects
- Can be performed manually or automatically
  - These tools provide an automatic way for highlighting possible coding errors
- Does require access to the code (or bytecode)
Our goal...

- Evaluate several automatic penetration testing tools and static analysis tools
  - In a controlled environment

Focus on two key measures of interest:

- Coverage
  - Portrays the percentage of existing vulnerabilities that are detected by a given tool

- False positives rate
  - Represents the number of reported vulnerabilities that in fact do not exist

Target only SQL Injection vulnerabilities

- Extremely relevant in Web Services
Steps

- **Preparation**
  - Select the penetration testers and static code analyzers
  - Select the Web Services to be considered

- **Execution**
  - Use the tools to identify potential vulnerabilities

- **Verification**
  - Perform manual verification to confirm that the vulnerabilities identified by the tools do exist
    - i.e., are not false positives

- **Analysis**
  - Analyze the results obtained and systematize the lessons learned
Web Services tested

- Eight Web Services
  - A total of 25 operations

- Four of the services are based on the TPC-App performance benchmark

- Four other services have been adapted from code publicly available on the Internet

- Implemented in Java and use a relational database
# Web Services characterization

<table>
<thead>
<tr>
<th>TPC-App</th>
<th>Service</th>
<th>Short Description</th>
<th>#Op</th>
<th>LoC</th>
<th>LoC/Op</th>
<th>Avg. C.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ProductDetail</td>
<td>Get details about a product</td>
<td>1</td>
<td>105</td>
<td>105,0</td>
<td>6,0</td>
</tr>
<tr>
<td></td>
<td>NewProducts</td>
<td>Add new product to the database</td>
<td>1</td>
<td>136</td>
<td>136,0</td>
<td>6,0</td>
</tr>
<tr>
<td></td>
<td>NewCustomer</td>
<td>Add new customer to the database</td>
<td>1</td>
<td>184</td>
<td>184,0</td>
<td>9,0</td>
</tr>
<tr>
<td></td>
<td>ChangePayment Method</td>
<td>Change customer’s payment method</td>
<td>1</td>
<td>97</td>
<td>97,0</td>
<td>11,0</td>
</tr>
<tr>
<td>Public-Code</td>
<td>JamesSmith</td>
<td>Manages personal data about students</td>
<td>5</td>
<td>270</td>
<td>54,0</td>
<td>6,0</td>
</tr>
<tr>
<td></td>
<td>PhoneDir</td>
<td>Phone book</td>
<td>5</td>
<td>132</td>
<td>26,4</td>
<td>2,8</td>
</tr>
<tr>
<td></td>
<td>Bank</td>
<td>Manages bank operations</td>
<td>5</td>
<td>175</td>
<td>35,0</td>
<td>3,4</td>
</tr>
<tr>
<td></td>
<td>Bank3</td>
<td>Manages bank operations (different from the Bank service)</td>
<td>6</td>
<td>377</td>
<td>62,8</td>
<td>9,0</td>
</tr>
</tbody>
</table>
Tools studied

- **Penetration testing**
  - HP WebInspect
  - IBM Rational AppScan
  - Acunetix Web Vulnerability Scanner
  - [Antunes 2009]

- **Static code analysis**
  - FindBugs
  - Yasca
  - IntelliJ IDEA

- **Decided not to mention the brand of the tools**
  - VS1, VS2, VS3, VS4 (without any order in particular)
  - SA1, SA2, SA3 (without any order in particular)
Tools and environment configuration

- **Penetration-testing**
  - Underlying database restored before each test
    - This avoids the cumulative effect of previous tests
    - Guarantees that all the tools started the service testing in a consistent state
  - If allowed by the testing tool, information about the domain of each parameter was provided
    - If the tool requires an exemplar invocation per operation, the exemplar respected the input domains of operation
    - All the tools in this situation used the same exemplar

- **Static code analysis**
  - Configured to fully analyze the services code
  - For the analyzers that use binary code, the deployment-ready version was used
Web Services manual inspection

- It is essential to correctly identify the vulnerabilities that exist in the services code.

- A team of experts was invited to review the source code looking for vulnerabilities.
  - False positives were eliminated by cross-checking the vulnerabilities identified by different people.

- A key difficulty is that different tools report (and count) vulnerabilities in different ways.
  - Penetration testing: a vulnerability for each vulnerable parameter.
  - Static analysis: a vulnerability for each vulnerable line in the service code.
## Vulnerabilities found

<table>
<thead>
<tr>
<th>Service</th>
<th>#Vuln. Inputs</th>
<th>#Vuln. Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductDetail</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NewProducts</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NewCustomer</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>ChangePaymentMethod</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>JamesSmith</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>PhoneDir</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Bank</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Bank3</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>61</strong></td>
<td><strong>28</strong></td>
</tr>
</tbody>
</table>
Penetration testing results

<table>
<thead>
<tr>
<th>Tool</th>
<th>% F. P.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS1</td>
<td>14.0%</td>
</tr>
<tr>
<td>VS2</td>
<td>4.0%</td>
</tr>
<tr>
<td>VS3</td>
<td>0.0%</td>
</tr>
<tr>
<td>VS4</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tool</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS1</td>
<td>50.8%</td>
</tr>
<tr>
<td>VS2</td>
<td>36.1%</td>
</tr>
<tr>
<td>VS3</td>
<td>9.8%</td>
</tr>
<tr>
<td>VS4</td>
<td>45.9%</td>
</tr>
</tbody>
</table>
Examples of penetration testing limitations

public void operation(String str) {
    try {
        String sql = "DELETE FROM table" + 
                     "WHERE id='' + str + "'";
        statement.executeUpdate(sql);
    } catch (SQLException se) {} }

public static String csvFromPath(String path) {
    // Open the file for reading
    BufferedReader br = new BufferedReader(new FileReader(path));
    // Read the file line by line
    String line;
    while ((line = br.readLine()) != null) {
        // Process the line here
    }
    // Close the file
    br.close();
    return null;
}

public String dumpDepositInfo(String str) {
    try {
        String path = "/\ DepositInfo/Deposit" + 
                      "[@accNum='" + str + "]";
        return csvFromPath(path);
    } catch (XPathException e) {} return null;
Static code analysis results

<table>
<thead>
<tr>
<th>Tool</th>
<th># False Positives</th>
<th># Vulnerable Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>7</td>
<td>23</td>
</tr>
<tr>
<td>SA2</td>
<td>10</td>
<td>28</td>
</tr>
<tr>
<td>SA3</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>Experts</td>
<td>28</td>
<td>28</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tool</th>
<th>% F. P.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>23.3%</td>
</tr>
<tr>
<td>SA2</td>
<td>26.3%</td>
</tr>
<tr>
<td>SA3</td>
<td>26.7%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tool</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>82.1%</td>
</tr>
<tr>
<td>SA2</td>
<td>100.0%</td>
</tr>
<tr>
<td>SA3</td>
<td>39.3%</td>
</tr>
</tbody>
</table>
Examples of static analysis limitations

public void operation(String str) {
    int i = Integer.parseInt(str);
    try {
        String sql = "DELETE FROM table" + 
                     "WHERE id='" + str + "'";
        statement.executeUpdate(sql);
    } catch (SQLException se) {}  
}

public String dumpDepositInfo(String str) {
    try {
        String path = "//DepositInfo/Deposit" + 
                      "[@accNum='" + str + "]";
        return csvFromPath(path);
    } catch (XPathException e) {}  
    return null;  
}
Penetration testing vs Static analysis (1)

Coverage

% Coverage

Marco Vieira

PRDC 2009, November 16-18, Shangai, China
Penetration testing vs Static analysis (2)

- False positives

% False Positives

![Bar chart showing % False Positives for VS1, VS2, VS3, VS4, SA1, SA2, and SA3]
Key observations

- The coverage of static code analysis is typically higher than of penetration testing

- False positives are a problem for both approaches
  - But have more impact in the case of static analysis;

- Different tools report different vulnerabilities in the same piece of code
  - Even tools implementing the same approach frequently

- Very poor results!
Conclusions

- The effectiveness of vulnerability detection tools is very low

How to improve penetration testing?
- Increase representativeness of the workload
- Guarantee high coverage
- Improve the attacks performed
- Improve the vulnerability detection algorithms

How to improve static analysis?
- Include new vulnerable code patterns

Merge penetration testing and static analysis?
Questions?