Detecting SQL Injection Vulnerabilities in Web Services

Nuno Antunes, Marco Vieira
{nmsa, mvieira}@dei.uc.pt
Outline

- Web Services
- Web Security Scanners
- New Approach for the Detection of SQL Injection Vulnerabilities in WS
- Experimental Evaluation
- Conclusions
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 of Web Services Environment](image)

Consumer

Registry (UDDI)

Provider 0

Provider 1

Provider 2

External Systems

WSDL

SOAP Request

SOAP Response
Examples of Vulnerabilities

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

    // Example SQL injection vulnerability
    sql = "SELECT * FROM users WHERE username=' OR 1=1 --" + login + "' AND password=' OR 1=1 --";
}
```

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

    // Example SQL injection vulnerability
    sql = "DELETE FROM table WHERE id=' OR '' = '';"
}
```
Web Security Scanners

- Easy and widely-used way to test applications searching vulnerabilities

- Use fuzzing techniques to attack applications
  - Penetration testing

- Perform thousands of tests in an automated way

- What is the effectiveness of these tools?
  - Can programmers rely on these tools?
Experimental Study

- Apply leading commercial scanners in public web services
- 300 Web Services tested
  - Randomly selected
- 4 Scanners used

Approach:
- Preparation: select services and scanners
- Execution: test the services using the scanners
- Verification: identify false positives
- Analysis: analysis and systematization of results
Scanners

HP WebInspect™

IBM Rational AppScan®

Acunetix Web Vulnerability Scanner
Overall results analysis

<table>
<thead>
<tr>
<th>Vulnerability Types</th>
<th>VS1.1</th>
<th>VS1.2</th>
<th>VS2</th>
<th>VS3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># Vuln.</td>
<td># WS</td>
<td># Vuln.</td>
<td># WS</td>
</tr>
<tr>
<td>SQL Injection</td>
<td>217</td>
<td>38</td>
<td>225</td>
<td>38</td>
</tr>
<tr>
<td>XPath Injection</td>
<td>10</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Code Execution</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Possible Parameter Based Buffer Overflow</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Possible Username or Password Disclosure</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Possible Server Path Disclosure</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>228</td>
<td>40</td>
<td>236</td>
<td>40</td>
</tr>
</tbody>
</table>
SQL Injection

- VS1.1: False Positives 87 (40%), Doubtful 14 (6.5%), Confirmed Vulnerabilities 116 (37%)
- VS1.2: False Positives 83 (37%), Doubtful 26 (11.6%), Confirmed Vulnerabilities 116 (25.7%)
- VS2: False Positives 8 (32%), Doubtful 17 (5), Confirmed Vulnerabilities 21 (14%)
- VS3: False Positives 9 (45%), Doubtful 5 (14%), Confirmed Vulnerabilities 21 (32%)

Marco Vieira
LADC 2009, September 01-04, João Pessoa, Brazil
Can we do better?

- Yes, we can! 😊
  - We propose a new approach to detect SQL Injection vulnerabilities in web services code

- Main improvements:
  - A representative workload to exercise the services and understand the expected behavior
  - A broader set of attacks
  - Well defined rules to analyze the service's responses
    - To improve coverage and remove false positives
  - Completely automatic
Execution Steps

1. **Prepare the tests**
   1.1. Gather information about the web service’s operations, call parameters, data types, and input domains
   1.2. Generate the workload

2. **Execute the tests**
   2.1. Execute the workload to understand the expected behavior of the service in the absence of attacks
   2.2. Perform the attacks to trigger faulty behaviors and disclose SQL Injection vulnerabilities

3. **Analyze the responses to detect and confirm the vulnerabilities**
Web service interfaces are described as a WSDL file.

This file is processed automatically to obtain:
- Operations
- Call parameters
- Data types

The valid values for each parameter (i.e., input domains) have to be provided by the user.
Prepare the Tests: Generate the Workload

Two options:
- User-defined workload
- Random workload

Random workload is generated automatically
- Generate test values for each input parameter
- Generate test calls for each operation
- Select test calls for each operation
  - It may be unfeasible to use a workload based on all the test calls generated (e.g., due to time constraints)
  - It is up to the user to specify the size of this subset
Execute the Tests: Configuration

Penetration Testing Tool

WE ALG

SOAP

Service Provider
Execute the Tests: Type of Attacks

**Examples:**

<table>
<thead>
<tr>
<th>SQL Injection Attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot; or l=1 --</td>
</tr>
<tr>
<td>&quot; or l=1 or &quot;&quot;=''</td>
</tr>
<tr>
<td>'or (EXISTS)</td>
</tr>
<tr>
<td>'or uname like '%'</td>
</tr>
<tr>
<td>'or userid like '%'</td>
</tr>
<tr>
<td>'or username like '%'</td>
</tr>
<tr>
<td>'UNION ALL SELECT</td>
</tr>
<tr>
<td>'UNION SELECT</td>
</tr>
<tr>
<td>char%2839%29%2b%28SELECT</td>
</tr>
<tr>
<td>&quot; or l=1 or &quot;&quot;=&quot;</td>
</tr>
<tr>
<td>' or ''='</td>
</tr>
</tbody>
</table>

**A total of 137 types**

- The list can be continuously improved
- Just add new attack patterns to a configuration file
Execute the Tests: Attacks Generation

- Mutation of the workload test calls
  - Valid values are replaced by malicious values

- Number of attacks can be extremely large, e.g.:
  - 3 operations with 5 parameters each
  - A workload with 25 test calls per operation
  - 137 attack types  52500 attacks

- The tool allows specifying the number of test calls to be used for the attack load generation
  - The original test calls are ranked based on their ability to help us detecting vulnerabilities
    - e.g. test calls that lead to valid web service responses (i.e., no error) are in the top of the list
Analyze the Responses

- W
  - Valid call
- A
  - Attack call
Experimental Evaluation

- Web services tested
  - 262 public web services

- Four steps:
  - Preparation: select a large set of web services.
  - Execution: use the vulnerability scanners to scan the services to identify potential vulnerabilities
  - Verification: perform manual testing to confirm that the vulnerabilities identified do exist
  - Analysis: analyze the results and compare the effectiveness of our tool to the commercial ones
Raw Results for Public Web Services

![Bar Chart]

- VS1.1: 147
- VS1.2: 156
- VS2: 21
- VS3: 26
- VS.WS: 79
What about false positives? (1)

- Manual checking

- Reported vulnerabilities are false positives if:
  - The error/answer obtained is related to a robustness problem and not to a SQL command
    - e.g., NumberFormatException
  - The error/value in response is not caused by the elements "injected" by the tool
    - i.e., the same problem occurs when the service is executed with valid inputs
What about false positives? (2)

Reported vulnerabilities are confirmed if:

- It is possible to observe that a SQL command was invalidated by the values "injected" by the tool
- The “injected” values lead to exceptions raised by the database server
- It is possible to access unauthorized services or web pages
  - e.g., by breaking the authentication process using SQL Injection

If none of these rules can be applied then a reported vulnerability is classified as doubtful.
Results for Public Web Services

<table>
<thead>
<tr>
<th>Scanner</th>
<th>% FP</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS1.1</td>
<td>63</td>
</tr>
<tr>
<td>VS1.2</td>
<td>55</td>
</tr>
<tr>
<td>V2</td>
<td>0</td>
</tr>
<tr>
<td>VS3</td>
<td>4</td>
</tr>
<tr>
<td>VS.WS</td>
<td>18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scanner</th>
<th>% Doubtful</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS1.1</td>
<td>9</td>
</tr>
<tr>
<td>VS1.2</td>
<td>16</td>
</tr>
<tr>
<td>V2</td>
<td>19</td>
</tr>
<tr>
<td>VS3</td>
<td>15</td>
</tr>
<tr>
<td>VS.WS</td>
<td>16</td>
</tr>
</tbody>
</table>
**Detection Coverage**

- Based on limited knowledge
  - Probably we don’t know all the existing vulnerabilities

<table>
<thead>
<tr>
<th>Scanner</th>
<th># Vul. Detected</th>
<th>Coverage %</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS1.1</td>
<td>47</td>
<td>84%</td>
</tr>
<tr>
<td>VS1.2</td>
<td>47</td>
<td>84%</td>
</tr>
<tr>
<td>VS2</td>
<td>17</td>
<td>30%</td>
</tr>
<tr>
<td>VS3</td>
<td>21</td>
<td>38%</td>
</tr>
<tr>
<td>VS.WS</td>
<td>52</td>
<td>93%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>56</strong></td>
<td><strong>100.0%</strong></td>
</tr>
</tbody>
</table>
Conclusions

- Results show that our approach achieves much better results than commercial tools
  - Concerning both coverage and false positives

- Our tool was able to detect vulnerabilities that were not detected by the commercial scanners
  - And, at the same time, was able to eliminate most of the false positives

- We show that it is possible to improve the state of the art in vulnerabilities detection
  - Future work includes extending our approach to different types of vulnerabilities and to other domains
Questions?

- http://eden.dei.uc.pt/~mvieira/
Results for TPC-App Web Services

- **Access to the source code**
  - Asked a team of security experts to find SQL Injection Vulnerabilities
  - Thorough code inspection and penetration tests

![Bar chart showing](attachment:chart.png)

- **Confirmed False Positives**
  - Security Team: 4
  - VS3: 6
  - VS.WS: 2

- **Confirmed Vulnerabilities**
  - Security Team: 22
  - VS3: 16
  - VS.WS: 18
False Positives and Coverage

- The team of security experts detected 26 vulnerabilities
  - 4 (~15%) were confirmed as false positives

- Our tool detected 18 vulnerabilities
  - Plus two false positives

- All the vulnerabilities detected by the tools were also identified by the security team

- The coverage of our tool was ~81%

- The false positives rate was 10%