Effective Detection of SQL/XPath Injection Vulnerabilities in Web Services

Nuno Antunes, Nuno Laranjeiro, Marco Vieira, Henrique Madeira
{nmsa, cnl, mvieira, henrique}@dei.uc.pt
Outline

- Web Security Scanners
- Effectiveness of vulnerability detection tools
- Our approach for the detection of SQL/XPath 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
Common vulnerabilities in Web Services

- [Vieira09] – 300 Public Web Services analyzed

- SQL Injection (149)
- Possible Server Path Disclosure (16)
- XPath Injection (10)
- Code Execution (1)
- Possible Parameter Based Buffer Overflow (1)
Vulnerability detection tool

- **Vulnerability Scanners**
  - Easy and widely-used way to test applications searching vulnerabilities
  - Use fuzzing techniques to attack applications
  - Avoid the repetitive and tedious task of doing hundreds or even thousands of tests by hand

- **Static code analyzers**
  - Analyze the code without actually executing it
  - The analysis varies depending on the tool sophistication
  - Provide a way for highlighting possible coding errors

**How effective are these tools?**
Experimental study

- Evaluate several automatic penetration testing tools and static analysis tools

- 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
Web Services used and tools assessed

- Eight Web Services tools
  - A total of 25 operations

- Penetration testing
  - HP WebInspect
  - IBM Rational AppScan
  - Acunetix Web Vulnerability Scanner

- Static code analysis
  - FindBugs
  - Yasca
  - IntelliJ IDEA
Penetration testing results

- **Tool** | % F. P.  
  |  
  | VS1 | 14.0%  
  | VS2 | 4.0%  
  | VS3 | 0.0%  

- **Tool** | Coverage  
  |  
  | VS1 | 50.8%  
  | VS2 | 36.1%  
  | VS3 | 9.8%
Static code analysis results

<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>
Key observations

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

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

- Different tools report different vulnerabilities in the same code

- Very poor results!
  - Can we do better?
Can we do better?

- Yes, we can! 😊
  - We propose a new approach to detect SQL/XPath Injection vulnerabilities in Web Services code

Steps:
1. Instrument the web service to intercept all SQL/XPath commands executed
2. Generate a workload
3. Execute the workload to learn SQL commands and XPath queries issued
4. Generate an attackload based on a large set of SQL Injection and XPath Injection attacks
5. Execute the attackload to detect vulnerabilities
Web Service instrumentation

- Based on Aspect-Oriented Programming (AOP)
Workload generation

1. Get relevant information from the WSDL file
   - Operations, parameters, datatypes

2. Generate test values for each input parameter
   - Random generation of a set of valid input values
   - The number of test values is defined by the user

3. Generate test calls for each operation
   - Generate a large set of calls for each operation
   - Sum of all combinations of the test values generated for all the parameters of each operation

4. Select test calls for each operation
   - Defined by the user
   - Determines the final size of the workload
1. Exercise the service by executing the workload

2. SQL and XPath commands are intercepted

3. Commands are parsed to remove the data variant part (if any)

4. A hash code is used to identify each command

5. Each hash signature is associated with a source code entry point

6. Workload coverage is analyzed
   - If not satisfactory, then more calls should be performed
Attackload

- Includes parameter values that attempt to perform SQL/XPath injection
- Attack types are based on the compilation of the types used by a large set of scanners
- Complemented based on practical experience and on information available in the literature

Attackload generation:
- Generate a new workload
- Malicious values are selectively inserted by applying the attack rules
### SQL Injection Attacks

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot; or 1=0 --</td>
</tr>
<tr>
<td>&quot; or 1=1 or &quot;&quot;=&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><code>char%2839%29%2b%28SELECT</code></td>
</tr>
<tr>
<td><code>&amp;quot; or 1=1 or </code>&quot;<code>&amp;quot;</code>=&quot;</td>
</tr>
<tr>
<td><code>&amp;apos; or </code>'<code>&amp;apos;</code>='</td>
</tr>
</tbody>
</table>
Vulnerabilities detection

1. Execute the attackload and perform security checks per each data access executed

2. SQL and XPath commands are intercepted and hashed

3. The calculated hash codes are compared to the values of the learned valid commands
   - For the code point at which the command was submitted

4. If hash code is NOT found then:
   - There is a vulnerability
   - The source code location was not learned correctly
Experimental evaluation

- Prototype tool to demonstrate the approach
  - Available at: http://eden.dei.uc.pt/~mvieira

- Experiments to assess its effectiveness
  - Detecting vulnerabilities in a set of Java-based Web Services coded by independent developers
  - Comparison with existing scanners and code analyzers

- Two key metrics were considered:
  - Detection coverage
    - Percentage of existing vulnerabilities detected by the tool
  - False positives rate
    - Percentage of vulnerabilities detected by the tool but that do not exist
Web Services tested

- Nine Web Services
  - A total of 28 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

- One service using XPath
Tools used

- Penetration testing
  - HP WebInspect
  - IBM Rational AppScan
  - Acunetix Web Vulnerability Scanner
  - VS.BB – [Antunes09]

- Static code analysis
  - FindBugs
  - Yasca
  - IntelliJ IDEA

- Decided not to mention the brand of the tools to assure neutrality
## Vulnerabilities found by manual inspection

<table>
<thead>
<tr>
<th>Service</th>
<th>Vulnerability Type</th>
<th>#Vuln. Inputs</th>
<th>#Vuln. Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductDetail</td>
<td>SQL Injection</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NewProducts</td>
<td>SQL Injection</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NewCustomer</td>
<td>SQL Injection</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>ChangePaymentMethod</td>
<td>SQL Injection</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>JamesSmith</td>
<td>SQL Injection</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>PhoneDir</td>
<td>SQL Injection</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Bank</td>
<td>SQL Injection</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Bank3</td>
<td>SQL Injection</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>XOperations</td>
<td>XPath Injection</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>65</strong></td>
<td><strong>32</strong></td>
</tr>
</tbody>
</table>
Comparing with penetration testing…

<table>
<thead>
<tr>
<th>Tool</th>
<th>% F. P.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS1</td>
<td>14%</td>
</tr>
<tr>
<td>VS2</td>
<td>4%</td>
</tr>
<tr>
<td>VS3</td>
<td>0%</td>
</tr>
<tr>
<td>VS.BB</td>
<td>0%</td>
</tr>
<tr>
<td>CIVS-WS</td>
<td>0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tool</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>VS1</td>
<td>48%</td>
</tr>
<tr>
<td>VS2</td>
<td>34%</td>
</tr>
<tr>
<td>VS3</td>
<td>9%</td>
</tr>
<tr>
<td>VS.BB</td>
<td>43%</td>
</tr>
<tr>
<td>CIVS-WS</td>
<td>100%</td>
</tr>
</tbody>
</table>
Intersections for penetration testing
Comparing with static code analysis...

![Graph showing # False Positives and # Vulnerable Lines for SA1, SA2, SA3, and CIVS-WS]

<table>
<thead>
<tr>
<th>Tool</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>71.9%</td>
</tr>
<tr>
<td>SA2</td>
<td>87.5%</td>
</tr>
<tr>
<td>SA3</td>
<td>34.4%</td>
</tr>
<tr>
<td>CIVS-WS</td>
<td>100.0%</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>
<tr>
<td>CIVS-WS</td>
<td>0.0%</td>
</tr>
</tbody>
</table>
Intersections for static code analysis
Conclusions and future work (1)

- Approach based on XPath and SQL commands learning and posterior detection
  - Based on the comparison the commands issued in the presence of attacks to the ones previously learned

- A tool fully implementing the proposed approach has been built
  - Allows automatic detection of SQL and XPath Injection vulnerabilities in Web Services

- An experimental evaluation has been performed over 9 Web Services
Conclusions and future work (2)

- Results show that the tool is quite effective
  - Achieved 100% coverage and 0% false positives rate in our experiments

- Some of the Web Services used in the evaluation are very complex
  - Suggests that the proposed approach can attain very good coverage even for high demanding scenarios

- Future:
  - Extend the approach to other types of vulnerabilities
  - Make the tool available to Web Service developers
Questions?

Marco Vieira
Center for Informatics and Systems
University of Coimbra
mvieira@dei.uc.pt