Preventing SQL Injection Attacks Using AMNESIA

William G.J. Halfond and Alessandro Orso
Georgia Institute of Technology

This work was partially supported by DHS contract FA8750-05-2-0214 and NSF awards CCR-0205422 and CCR-0209322 to Georgia Tech.
SQL Injection Attacks

- David Aucsmith (CTO of Security and Business Unit, Microsoft) defined SQLIA as one of the most serious threats to web apps
- Open Web Application Security Project (OWASP) lists SQLIA in its top ten most critical web application security vulnerabilities
- Successful attacks on Guess Inc., Travelocity, FTD.com, Tower Records, RIAA…
Presentation Outline

• Motivation
• Background Info.
• AMNESIA
• Demonstration
• Evaluation Overview
• Summary
String queryString = "SELECT info FROM userTable WHERE ";
if (!(login.equals("")) && !(pin.equals(""))) {
    queryString += "login='' + login + '' AND pin='' + pin ;
} else {
    queryString += "login='guest''; 
}
ResultSet tempSet = stmt.execute(queryString);
Attack Scenario

String queryString = "SELECT info FROM userTable WHERE ";
if ((! login.equals("")) && (! pin.equals(""))) {
    queryString += "login=" + login + " AND pin=" + pin ;
} else {
    queryString+="login='guest'";
}
ResultSet tempSet = stmt.execute(queryString);

Normal Usage
¬User submits login “doe” and pin “123”
¬SELECT info FROM users WHERE login= `doe’ AND pin= 123
Attack Scenario

String queryString = "SELECT info FROM userTable WHERE ";
if ((! login.equals("")) && (! pin.equals(""))) {
    queryString += "login='" + login + "' AND pin='" + pin + ""
} else {
    queryString+="login='guest'"
}
ResultSet tempSet = stmt.execute(queryString);

Malicious Usage
¬Attacker submits “user’ -- ” and pin of “0"
¬SELECT info FROM users WHERE login=‘user’ -- ’ AND pin=0
Many types of SQLIA

<table>
<thead>
<tr>
<th>Types</th>
<th>Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Piggy-backed Queries</td>
<td>• User input</td>
</tr>
<tr>
<td>• Tautologies</td>
<td>• Cookies</td>
</tr>
<tr>
<td>• Alternate Encodings</td>
<td>• Server variables</td>
</tr>
<tr>
<td>• Inference</td>
<td>• Second-order</td>
</tr>
<tr>
<td>• Illegal/Logically Incorrect Queries</td>
<td>• …</td>
</tr>
<tr>
<td>• Union Query</td>
<td></td>
</tr>
<tr>
<td>• Stored Procedures</td>
<td></td>
</tr>
</tbody>
</table>
AMNESIA

Basic Insights

1. Code contains enough information to accurately model all legitimate queries.
2. A SQL Injection Attack will violate the predicted model.

Solution:

Static analysis => build query models
Runtime analysis => enforce models
Overview of AMNESIA

1. Identify all hotspots.
2. Build SQL query models for each hotspot.
3. Instrument hotspots.
4. Monitor application at runtime.
1 – Identify Hotspots

Scan application code to identify hotspots.

```java
String queryString = "SELECT info FROM userTable WHERE ";
if (! login.equals("")) && (! pin.equals("")) {  
    queryString += "login='" + login + "' AND pin='" + pin;
} else {
    queryString+="login='guest'";
}
ResultSet tempSet = stmt.execute(queryString);
```

Hotspot
2 – Build SQL Query Model

1. Use Java String Analysis\(^1\) to construct character-level automata
2. Parse automata to group characters into SQL tokens

```sql
SELECT info FROM userTable WHERE login = 'guest' AND pin = \beta
```
Wrap each hotspot with call to monitor.

```java
String queryString = "SELECT info FROM userTable WHERE ";
if (!(login.equals("")) && (!(pin.equals(""))) ) {
    queryString += "login='" + login + "' AND pin='" + pin + ""
} else {
    queryString+="login='guest'";
}

if (monitor.accepts (hotspotID, queryString) ) {
    ResultSet tempSet = stmt.execute(queryString);
}
```
4 – Runtime Monitoring

Check queries against SQL query model.

Normal Usage:

```
SELECT info FROM userTable WHERE login = 'guest' AND pin = 'β'
SELECT info FROM userTable WHERE login = 'doe' AND pin = 123
```
Check queries against SQL query model.

Malicious Usage:

```
SELECT info FROM userTable WHERE login = 'guest' AND pin = 0
```

```
SELECT info FROM userTable WHERE login = 'β' AND pin = β
```

AMNESIA Implementation
AMNESIA Demonstration

• Attacking a commercial application:
  • Evade login protection
  • Change contents of the database – “Special sale price”
• Blocking attacks with AMNESIA
• Examine SQL query models
Evaluation: Research Questions

**RQ1**: What percentage of attacks can our technique detect and prevent that would otherwise go undetected and reach the database?

**RQ2**: How much overhead does our technique impose on web applications at runtime?

**RQ3**: What percentage of legitimate accesses does our technique prevent from reaching the database?
## Evaluation: Experiment Setup

<table>
<thead>
<tr>
<th>Subject</th>
<th>LOC</th>
<th>Hotspots</th>
<th>Average Automata size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checkers</td>
<td>5,421</td>
<td>5</td>
<td>289 (772)</td>
</tr>
<tr>
<td>Office Talk</td>
<td>4,543</td>
<td>40</td>
<td>40 (167)</td>
</tr>
<tr>
<td>Employee Directory</td>
<td>5,658</td>
<td>23</td>
<td>107 (952)</td>
</tr>
<tr>
<td>Bookstore</td>
<td>16,959</td>
<td>71</td>
<td>159 (5,269)</td>
</tr>
<tr>
<td>Events</td>
<td>7,242</td>
<td>31</td>
<td>77 (550)</td>
</tr>
<tr>
<td>Classifieds</td>
<td>10,949</td>
<td>34</td>
<td>91 (799)</td>
</tr>
<tr>
<td>Portal</td>
<td>16,453</td>
<td>67</td>
<td>117 (1,187)</td>
</tr>
</tbody>
</table>

- Applications are a mix of commercial (5) and student projects (2)
- Attacks and legitimate inputs developed *independently*
- Attack inputs represent broad range of exploits
## Evaluation Results: RQ1

<table>
<thead>
<tr>
<th>Subject</th>
<th>Unsuccessful</th>
<th>Successful</th>
<th>Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checkers</td>
<td>1195</td>
<td>248</td>
<td>248 (100%)</td>
</tr>
<tr>
<td>Office Talk</td>
<td>598</td>
<td>160</td>
<td>160 (100%)</td>
</tr>
<tr>
<td>Employee Directory</td>
<td>413</td>
<td>280</td>
<td>280 (100%)</td>
</tr>
<tr>
<td>Bookstore</td>
<td>1028</td>
<td>182</td>
<td>182 (100%)</td>
</tr>
<tr>
<td>Events</td>
<td>875</td>
<td>260</td>
<td>260 (100%)</td>
</tr>
<tr>
<td>Classifieds</td>
<td>823</td>
<td>200</td>
<td>200 (100%)</td>
</tr>
<tr>
<td>Portal</td>
<td>880</td>
<td>140</td>
<td>140 (100%)</td>
</tr>
</tbody>
</table>

⇒ No false negatives

⇒ Unsuccessful attacks = filtered by application
Evaluation Results: RQ2 & RQ3

• Runtime Overhead
  • Less than 1ms.
  • Insignificant compared to cost of network/database access

• No false positives
  • No legitimate input was flagged as SQLIA
Conclusions & Future Work

- AMNESIA detects and prevents SQLIAs by using static analysis and runtime monitoring
  - Builds models of expected legitimate queries
  - At runtime, ensure all generated queries match model
- In our evaluation
  - No false positives
  - No false negatives
- Future work => address limitations
  - Imprecision in static analysis
  - External trusted input