Revisiting SQL Injection
Will we ever get it right?

Michael Sutton, Security Evangelist
Overview

Background
• What it is?
• How are we doing?

Web 2.0
• SQL injection meets AJAX

Fuggle
• SQL Injection meets Google meets fuzzing

Conclusion
Attack Scenario #1
Good ‘ol SQL Injection
SQL Injection

Unfiltered User Input

- Attacker can influence or rewrite a backend SQL query

Client Request

POST /SPIWare/Login.aspx HTTP/1.1
Host: localhost
txtEmail=msutton@spidynamics.com&txtPassword=password

SQL Query

SELECT * FROM Customers WHERE Email = '{0}' and Password = '{1}'
{txtEmail.Text}, {txtPassword.Text}
SQL Injection

**Cause**

- Relational Database
- Injected SQL concatenated into SQL statement
  
  *Example:* `SELECT * FROM users` where `user = '' HAVING 1=1` --

**Definition**

- #3 SQL result/error returned
  
  *Example:* Column 'Users.UserID' is invalid in the select list...

**Impact**

- #1 Inject SQL
  
  *Example:* `\ HAVING 1=1` --

- #4 Web page response returned to attacker
  
  *Example:* HTTP 500 Internal Server Error
  
  "An error has occurred..."

**Demo**

- Vulnerable Web Server
- Attacker
What is SQL Injection?

Verbose SQL Injection
- Server provides detailed error messages
- Error messages are leveraged by attacker to gain insight into database structure

Blind SQL Injection
- Database error messages are suppressed
- Attacker must alter approach and only make queries with Boolean outcomes (yes/no questions)
## Cause vs. Blind SQL Injection

<table>
<thead>
<tr>
<th>Cause</th>
<th>Definition</th>
<th>Impact</th>
<th>Demo</th>
</tr>
</thead>
</table>

**Server Error in '/Hackware' Application.**

*Column 'Customers.CustomerID' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.*

**Description:** An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

---

**Whoops…an error has occurred!**
What damage can be done?

**Confidentiality**
- SELECT
  - Read from application and system tables

**Integrity**
- DELETE, INSERT, DROP
  - Alter existing data or inject new data

**Authentication Bypass**
- E.g. `' OR 1=1 --`

**System Compromise**
- Stored procedures
- Extended stored procedures
SQL Injection Demo

Sign In

If you don't have an account, click here to create one.

Click here if you've forgotten your password.

Enter your Information

- Email
- Address:
- Password:

[Sign-In]

(Do NOT use this feature if you are using a public computer or if you share your computer with other users. Please note that this feature is local to the current browser and computer that you are using now.)
Prevention

Input Validation
- Always ensure that you receive what you expect
- White list vs. black list
- Validate type, length, format, and range
  - Regular expressions - http://regexlib.com
- Escape special characters
- Always validate on the server side!

Leverage Frameworks
- Parameterized Queries
- ‘Point and Click’ query generation

ACLs
- Execute queries with least privilege
Mitre CVE Statistics

2001

- XSS
- Buffer Overflow
- SQL Injection
- Directory Traversal
- PHP File Inclusion
- Information:
- DoS Malformed:
- Symbolic Link
- Format String
- Cryptographic Error
WASC Statistics - 2006

Percentage of websites vulnerable by class (Top 5):

- Cross-Site Scripting: 35.57%
- SQL Injection: 26.38%
- Information Leakage: 15.70%
- HTTP Response Splitting: 9.76%
- Path Traversal: 4.30%
- Other: 1.19%
- Other: 1.19%
WASC Statistics – 2007 (1H)

% of Vuln. Sites

- Cross Site Scripting: 26.49%
- Information Leakage: 10.34%
- SQL Injection: 8.11%
- Predictable Resource Location: 5.58%
- Content Spoofing: 1.11%
Attack Scenario #2
SQL Injection via AJAX
What is Web 2.0?

- **Web 2.0 is the business revolution in the computer industry caused by the move to the internet as platform, and an attempt to understand the rules for success on that new platform.**

  *Tim O’Reilly*

- **Web 2.0...refers to a perceived second-generation of Web based communities and hosted services — such as social networking sites, wikis and folksonomies — that facilitate collaboration and sharing between users.**

  *Wikipedia*
Web 2.0 My Definition

**Web 1.0**
- Incomplete pages were shameful
- “Please come back later when we’re ready”

**Web 2.0**
- Incomplete pages are a feature!
- “Stick around and help us improve the site”

Same Vulnerabilities
- Additional Input Vectors
- More Complexity
Input Vectors

Web 1.0

Web 2.0

RSS/Atom

SOAP

XMLHttpRequest

HTML
AJAX

What does AJAX change?

• Business logic is dispersed among multiple client side files/functions
• Requests are made in the background without user intervention but are just as susceptible to attack

What doesn’t AJAX change?

• AJAX does not create new vulnerabilities

How does AJAX affect security?

• Increased surface area
  • More business logic is exposed
  • New input vectors are exposed
  • Security tools must understand the XHR objects and their syntax in order to identify input vectors
What is AJAX?

**Asynchronous**

- Requests are initiated in the background

**JavaScript**

- JavaScript instantiates the XmlHttpRequest object and generates the requests

**and XML**

- This is a misnomer as AJAX frameworks commonly employ alternate data interchange formats
  - JSON - Atlas
  - Serialized Java - Google Web Toolkit
  - HTML
  - XML
What is AJAX?

Multiple frameworks
- Prototype (http://www.prototypejs.org/)
- Script.aculo.us
- Dojo (http://dojotoolkit.org/)
- ASP.Net AJAX (http://ajax.asp.net/)
- Etc.

Multiple browser objects
- Internet Explorer
  - IE6 - XMLHttpRequest ActiveX control
  - IE7 – XMLHttpRequest native script object
- Firefox
  - XMLHttpRequest object
Classic Web Apps

- **Cause**
- **Definition**
- **Impact**
- **Demo**

**Diagram:**
- **Client**
  - User activity
  - Data transmission
  - System processing

- **Server**
  - User activity
  - Data transmission
  - System processing

**Classic web application model (synchronous)**
AJAX Web Apps

Ajax web application model (asynchronous)
FireBug

<table>
<thead>
<tr>
<th>File</th>
<th>URL</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>bounds.txt</td>
<td>maps.google.com</td>
<td>208 b</td>
</tr>
<tr>
<td>gen_204 (204)</td>
<td>maps.google.com</td>
<td>0 b</td>
</tr>
</tbody>
</table>

### Params

- **cad**: src:link,cid:11136651394417860985,latlong:36091817.-115173717
- **cd**: 1
- **ct**: miw basics
- **ei**: LUJ5Rt_PFKiKiA0xyKjXdw
- **iwc**: Ch7odHRwOi8vd3d3Lm1hbmrhbGF5YmF5LmNvbS8QBBAGEAkQBRgy
- **oi**: miw
- **sa**: T
- **sig2**: IvfJB5CDyJQoCwcClvzpg

<table>
<thead>
<tr>
<th>File</th>
<th>URL</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>gen_204 (204)</td>
<td>maps.google.com</td>
<td>0 b</td>
</tr>
</tbody>
</table>

3 requests | 208 b
What damage can be done?

Same old story

- Any classic web application vulnerabilities can be exploited through AJAX
- SQL injection
- XSS
- Etc.

Risk

- In a rush to implement the latest Web 2.0 technologies, companies are ignoring security
**How Not to Implement AJAX - BlinkList**

```plaintext
| Response Headers          | Value                                                                 |
|--------------------------|                                                                     |
| Date                     | Wed, 28 Feb 2007 03:46:36 GMT                                     |
| Server                   | Apache/2.0.52 (CentOS)                                            |
| Last-Modified            | Fri, 29 Dec 2006 14:00:55 GMT                                     |
| Etag                     | "b0068-23-42sbebl145d3c0"                                         |
| Accept-Ranges            | bytes                                                                |
| Cache-Control            | max-age=2592000                                                    |
| Expires                  | Fri, 30 Mar 2007 03:46:36 GMT                                     |
| Content-Type             | image/gif                                                           |

**Request Headers**

- **Host**: www.blinklist.com
- **User-Agent**: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.10) Gecko/20070216 Firefox/1.5.0.10
- **Accept**: image/png,"/*;q=0.5
- **Accept-Language**: en-us;en;q=0.5
- **Accept-Encoding**: gzip,deflate
- **Accept-Charset**: ISO-8859-1,utf-8;q=0.7,*;q=0.7
- **Keep-Alive**: 300
- **Connection**: keep-alive
- **Referer**: http://www.blinklist.com/tag/ajax/
- **Cookie**: cidp1094=12242826; PHPSESSID=1fajl4u9ehurq5a78p8n7fdo5; cidp147=11483711
```
select usertag.name from usertag where usertag.userid = order by usertag.name
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'order by usertag.name' at line 1

Warning  implode() [function.implode]: Bad arguments in /home/blinklis/public_html/Userpage/Startpage/getmytag.ax.php on line 13
How Not to Implement AJAX - BlinkList

- Verbose SQL errors
  - Multiple

- XSS
  - Ability to inject client side script

- Exposed functionality
  - Web based email

- Directory browsing
  - Access to restricted files
AJAX Demo

Sign In

If you don't have an account, click here to create one.

Click here if you've forgotten your password.

Enter your Information

Email: [Enter email]
Address: [Enter address]
Password: [Enter password]

[Sign In]

(Do NOT use this feature if you are using a public computer or if you share your computer with other users. Please note that this feature is local to the current browser and computer that you are using now.)
Fuzzing
Using
Google
Gets
Low hanging fruit
Easily
FUGGLE

Fuzzing

Using

Google

Gets

Low hanging fruit

Easily

Fuggle™

Google gets low hanging fruit easily.
A Russian hackers broke into a Rhode Island government Web site and allegedly stole credit card data from individuals who have done business online with state agencies.

The story was first reported by The Providence Journal this morning and comes two days after state and local government officials released national surveys indicating they need more cybersecurity guidance and help in strengthening their systems.
Fuggle Fuzzing Phases

- Identify Input
- Identify Targets
- Generate Fuzzed Data
- Execute Fuzzed Data
- Monitor for Exceptions
- Determine Exploitability
# Fuggle vs. Google Hacking

<table>
<thead>
<tr>
<th>Fuggle</th>
<th>Google Hacking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus on input</td>
<td>Focus on output</td>
</tr>
<tr>
<td><em>e.g. URI parameters</em></td>
<td><em>e.g. page content</em></td>
</tr>
<tr>
<td>Identifying targets for further testing</td>
<td>Identifying pages using vulnerable 3rd party apps or leaking confidential information</td>
</tr>
<tr>
<td>Flexible search terms</td>
<td>Fixed signature based searches</td>
</tr>
<tr>
<td><em>e.g. inurl:&quot;id=10&quot;</em></td>
<td><em>e.g. intitle:index.of &quot;parent directory&quot;</em></td>
</tr>
<tr>
<td>Custom vulnerabilities</td>
<td>Known vulnerabilities</td>
</tr>
</tbody>
</table>
Fuggle Prerequisites

**Fuzz Variables**

- Input vectors must be indexed by Google and accessible via search operators
  - ✔ - Title
  - ✔ - Displayed page content
  - ✔ - URI
  - ✗ - Request/response headers
  - ✗ - Page source code

**Limitations**

- Effectively limits using Fuggle to pages using GET method
- Input vectors indexed in URL
Fuggle Threat

How can Fuggle be abused?

- Indiscriminate web application hacking
- Vulnerability scanning for self propagating worms / web application worms
Fuggle SQL Injection – Identify Input

**Input**

- User supplied values concatenated into SQL queries

```sql
www.example.com?id=10
SELECT product from products WHERE id=10;
```

**Goal**

- Identify pages with verbose SQL errors
Fuggle SQL Injection – Identify Targets

- **Search Term**
  - inurl:"id=10"

- **Targets**
  - Retail stores
    - E.g. Product catalog
  - Informational sites
    - E.g. News archive

- **Search results**
  - Results 1 - 10 of about 2,010,000 for inurl:"id=10". (0.05 seconds)

- **Cleanse results**
  - Remove URLs w/out “id=10”
  - Remove duplicate results from single domain
Fuggle SQL Injection – Generate Data

Goal

- Identify pages with verbose SQL errors

Fuzz data

- Verbose SQL injection
  - id='10
- Blind SQL injection
  - id=10 OR 1=1
- Comment remainder of query
  - id='10--
- Encode query
  - id=%2710

Identify Input

Identify Targets

Generate Fuzzed Data

Execute Fuzzed Data

Monitor for Exceptions

Determine Exploitability
Submit queries

Capture responses
- Raw response
- Headers
- HTML source code
- HTML Status codes

Associate requests with responses

Archive for automated and manual review
Fuggle SQL Injection – Monitor Exceptions

Identify Input
Identify Targets
Generate Fuzzed Data
Execute Fuzzed Data
Monitor for Exceptions
Determine Exploitability
SQL Injection – Exploitability

Confidentiality
- SELECT

Integrity
- DROP
- INSERT
- DELETE

System compromise
- Stored procedures
- Extended stored procedures

Identify Input
Identify Targets
Generate Fuzzed Data
Execute Fuzzed Data
Monitor for Exceptions
Determine Exploitability
<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial population of URLs</td>
<td>1,000</td>
</tr>
<tr>
<td>Population after removal of duplicate servers</td>
<td>732</td>
</tr>
<tr>
<td>Population after removal of failed requests</td>
<td>708</td>
</tr>
<tr>
<td>Total number of verbose SQL errors</td>
<td>80</td>
</tr>
<tr>
<td>Percentage of sample web sites potentially vulnerable to SQL injection attacks</td>
<td>11.3%</td>
</tr>
</tbody>
</table>
Questions

Michael Sutton, Security Evangelist
http://portal.spidynamics.com/blogs/msutton
Michael.Sutton@hp.com