

Avoiding SQL Injection: Don't Let a Stranger "Shoot You Up"



Presented by: John Jay King
Download this paper from: <http://www.kingtraining.com>



- Learn how SQL Injection occurs and why
- Code program SQL to avoid SQL Injection
- Write more-secure software



- Understanding SQL Injection
 - How SQL Injection occurs
 - What can people do with SQL Injection?
- Stopping SQL Injection
- Coding techniques to avoid SQL Injection



- John King – Partner. King Training Resources
- Oracle Ace Director 
- Member Oak Table Network 
- Providing training to Oracle and IT community for over 25 years – <http://www.kingtraining.com>
- “Techie” who knows Oracle, ADF, SQL, Java, and PL/SQL pretty well (along with many other topics)
- Member of ODTUG (Oracle Development Tools User Group) Board of Directors (until Jan 1!)



- SQL Injection is the act of “adding to” SQL unbeknownst to the application
 - Various techniques are widely publicized enabling patient and/or automated efforts
 - SQL queries may be modified in many environments
 - Additional SQL may be executed in some environments



- Web/Client-Server applications request user input; the user input is then added to SQL in the code

If the code in the application is sloppy; a patient attacker can add to the SQL being executed

Or, “Inject” code into the SQL



- Query modifications (all databases)
 - Adding/modifying clauses
 - Adding UNION / UNION ALL
- Column/Table interrogation
- Adding / Changing / Deleting data rows
- CREATE TABLE, DROP TABLE, etc...
- In some cases; perform admin functions like SHUTDOWN and even operating system commands



- According to the OWASP (Open Web Security Project) site's

2013 “[OWASP Top Ten Project](#)” list

#1 Security Risk

[A1: Injection](#)

#1 for several years!



- According to the 2011 Common Weakness Enumeration site's ["2011 CWE/SANS Top 25 Most Dangerous Software Errors"](#) list

#1 with weakness score of 93.8 out of 100

[CWE-89](#) Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')



- [Wikipedia's entry for SQL Injection](#) provides a list of **publicized** SQL Injection attacks including these (see site for more)
 - On 5 February 2011 HBGary, a technology security firm, was broken into by Anonym
 - On March 27, 2011 [mysql.com](#), the official homepage for MySQL, was compromis
 - On April 11, 2011, Barracuda Networks was compromised using an SQL injection fla
 - Over a period of 4 hours on Wednesday April 27, 2011 an automated SQL injection : 8,000 random accounts of the 9,000 active and 90,000 old or inactive accounts.^{[37][3}
 - On June 1, 2011, "hacktivists" of the group Lulzsec were accused of using SQLI to s personal information of a million users.^{[40][41]}
 - In June, 2011, PBS was hacked, mostly likely through use of SQL injection. The full
 - On June 27, 2011, Lady Gaga's website was hacked by a group of US cyber attacke content database dump from [www.ladygaga.co.uk](#) and a section of email, first name vulnerability for her website was recently posted on a hacker forum website, where a compromised, the blog implies that Lady Gaga fans are most likely receiving fraudul
 - In August, 2011, Hacker Steals User Records From Nokia Developer Site using "SQ
 - In September, 2011, Turkish Hackers accessed NetNames DNS records and change UPS, Acer, Vodafone.com) to a site set up by them, and then taking responsibility fi
 - In October, 2011, Malaysian Hacker, Phiber Optik managed to extract data from ww company within minutes and claiming to have used SQL Injection.



- SQL Injection may be performed on all major databases:
 - Oracle
 - DB2
 - MySQL
 - SQL Server
 - PostgreSQL
 - and more...
- SQL Injection is more-difficult in advanced products like Oracle and DB2

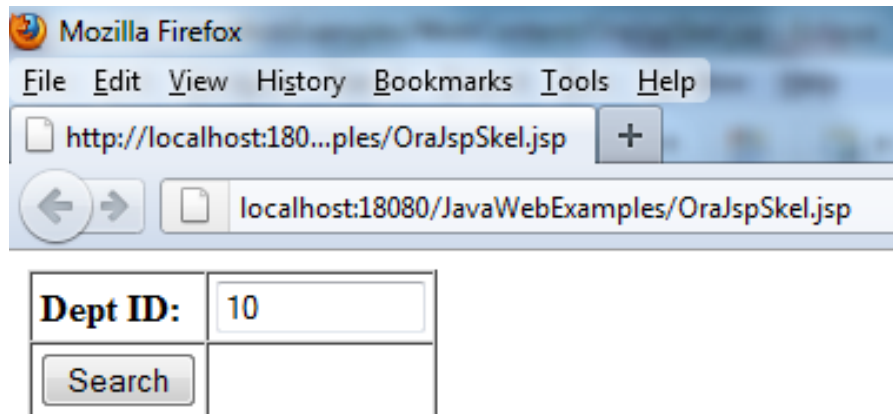


- Any application accepting user input that is inserted into SQL; especially when strings are concatenated into WHERE or HAVING (but possible in other ways....)
- Simple tools like PHP and ASP are easier to attack
- Programmatic environments like Java EE and .NET offer fewer vulnerabilities but can still be attacked
(so can poorly-written ADF and APEX)

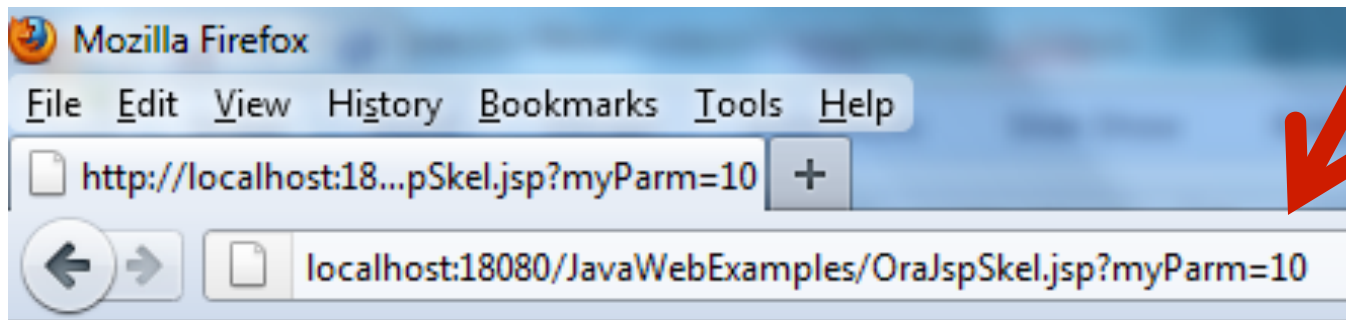


- First, a sloppy SQL statement opens the door using user input to complete code:

```
String sql =
    "SELECT empno, ename, hiredate, sal, deptno "
    + " FROM emp "
    + " WHERE deptno = " + myParm + " ";
```



- Application shows data; note how use of parameter is shown in URL (myParm=10)



Dept ID:	<input type="text" value="10"/>			
<input type="button" value="Search"/>				
Dept No	Name	Emp No	Salary	Hire date
10	CLARK	7782	2450	1981-06-09 00:00:00
10	KING	7839	5000	1981-11-17 00:00:00
10	MILLER	7934	1300	1982-01-23 00:00:00



- What happens if I modify the parameter and resubmit?
 - First, let's try
myParm=10 or 1=1 --
 - No luck, how about...
myParm=10' or 1=1 --
 - Jackpot! All rows shown!

localhost:18080/JavaWebExamples/OraJspSkel.jsp?myParm=10' or 1=1 --

Dept ID:	10' or 1=1 --			
Search				
Dept No	Name	Emp No	Salary	Hire date
20	SMITH	7369	800	1980-12-17 00:00:00
30	ALLEN	7499	1600	1981-02-20 00:00:00
30	WARD	7521	1250	1981-02-22 00:00:00
20	JONES	7566	2975	1981-04-02 00:00:00
30	MARTIN	7654	1250	1981-09-28 00:00:00
30	BLAKE	7698	2850	1981-05-01 00:00:00
10	CLARK	7782	2450	1981-06-09 00:00:00
20	SCOTT	7788	3000	1982-12-09 00:00:00
10	KING	7839	5000	1981-11-17 00:00:00
30	TURNER	7844	1500	1981-09-08 00:00:00
20	ADAMS	7876	1100	1983-01-12 00:00:00
30	JAMES	7900	950	1981-12-03 00:00:00
20	FORD	7902	3000	1981-12-03 00:00:00
10	MILLER	7934	1300	1982-01-23 00:00:00



- The SQL was coded as follows:

```
WHERE deptno = ' ' + myParm + ' ' "
```

- The “user” entered:

```
10' or 1=1 --
```

- The SQL executed was:

```
WHERE deptno = '10' or 1=1 --
```

- The first apostrophe, closed the input string
- “or 1=1” is always true (fetching all rows)
- The two hyphens “--” commented the rest of the line



- Once an attacker has noticed that your SQL is vulnerable; a series of tests can be made to discover:
 - How many columns in current query
 - Order of columns in current query
 - Types of data in current query
- Once a little information is gained, another series of test/probes can discover
 - Type of database
 - Access allowed to application user



- Add an order by using column numbers to determine how many columns in the query
- Observe output while sorting to see order of columns, attempt to deduce data type
- Add “UNION ALL” or “UNION” and a dummy query to be sure of original query
- Add “UNION ALL” of some important information to determine access rights and database type



- Here are some injections to the example code that worked
 - Prove SQL can be injected
`' or 1=1 --`
 - Determine number of columns
`' or 1=1 order by 6 --`
 - Determine sequence/type of columns
`' or 1=1 order by 1 desc --`
 - What Query Looks like (using screen names)
`-- select column data is:
-- emp no, name, hire date, salary dept no`



- The following statements list the users in the test Oracle database (first) and their role privileges (second)
- Knowledgeable persons can get similar information from any product provided that the application user has rights...

```
1' union all select user_id,username || '/' ||  
password,created,0,0 from dba_users -
```

```
1' union all select 0,granted_role || '-' ||  
grantee,null,0,0 from dba_role_privs order by 2 --
```



- Some application code executes SQL in a manner that would allow multiple statement to be executed
(usually delimited by a semi-colon “;”)
- If the application user is empowered can create/drop tables, run admin functions, etc...

```
myParm=1 ' ; DELETE CUSTOMER_MASTER; --
```



- The keys to mitigating SQL injection are:
 - Controlling security profile of running code
 - Limiting opportunities for attack
 - Carefully editing and validating user input



- Some SQL Injection exploits and impacts are possible simply because the userid used in an application is too powerful
 - Application userids should not have administrative authority of any kind
 - Application userids should not have authority to see data that is not part of the application
 - Applications that must insert/update/delete data should use Stored Procedures so that the application user need not be empowered



- Use Parameterized API's such as Java JDBC Prepared Statements

```
String sql    = "SELECT empno, ename, hiredate, sal,  
                +      " deptno from emp "  
                +      " WHERE deptno = ? ";
```

```
PreparedStatement stmt = conn.prepareStatement(sql) ;  
stmt.setInt(1,myIntParm) ;  
ResultSet rset = stmt.executeQuery() ;
```




- Just because an application is using parameterized input and/or Stored Procedures does not mean you are safe !
- Unchecked input can still wreak havoc





- SQL Injection may be avoided by editing and validating input
 - Length of data
 - Correct type of data
 - Unwanted / Undesired Characters
 - Unwanted / Undesired String Combinations



- Validate input data to make sure length is within expected bounds



- Do not use “string”-type data for input except where “string”-type is required;

Most SQLs will simply convert the data of the column being tested for comparison



- In cases where lengthy “string”-type data is required for input; check to see if unwanted characters are present
 - Semi-colons, colons, comparison operators
 - Quotes and Apostrophes (but what if it’s a name field and you want to allow O’ Brian?); look for repetitions of quotes and apostrophes
 - Double-hyphen comments “--”



- Finally, some it's not a bad idea to subject input values to some type of “reality check” to make sure values entered are reasonable for the input field and not just an experimental stab by some intruder



- Dumping SQL error messages onto the user's screen can be confusing to the customer; and it can provide information to an attacker!
- Provide messages about the values expected; do not add superfluous information about column names, data types, etc...



- Beware the “safety” of the escape character ('\'); this can be spoofed too!
- OWASP has a library of routines that might help (not a panacea, but helpful)
<https://www.owasp.org/index.php/ESAPI>
 - Look for “Escaping” and “Whitelist” APIs



- If lengthy “string”-type data is required; check for
 - SQL keywords
 - Administrative keywords
 - Use of SQL-type operators and characters



- OWASP has a resource page to help with SQL injection

https://www.owasp.org/index.php/Top_10_2013-A1



- I’m all for free help, here’s an excellent cheat sheet provided by OWASP

https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet





- The CWE also provides a list of resources for mitigating SQL Injection

<http://cwe.mitre.org/top25/index.html#CWE-89>

1
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Summary

Weakness Prevalence	High	Consequences	Data loss, Security bypass
Remediation Cost	Low	Ease of Detection	Easy
Attack Frequency	Often	Attacker Awareness	High

Discussion

These days, it seems as if software is all about the data: getting it into the database, pulling it from the database, massaging information, and sending it elsewhere for fun and profit. If attackers can influence the SQL that you use to communicate with database, then suddenly all your fun and profit belongs to them. If you use SQL queries in security controls such as authenticating attackers could alter the logic of those queries to bypass security. They could modify the queries to steal, corrupt, or otherwise underlying data. They'll even steal data one byte at a time if they have to, and they have the patience and know-how to do so. SQL injection was responsible for the compromises of many high-profile organizations, including Sony Pictures, PBS, MySQL.com, company HBGary Federal, and many others.

[Technical Details](#) |
 [Code Examples](#) |
 [Detection Methods](#) |
 [References](#)

Prevention and Mitigations

Architecture and Design

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, consider using persistence layers such as Hibernate or Enterprise Java Beans, which can provide significant protection against SQL properly.

Architecture and Design

If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to pre-



- OWASP (Open Web Application Security Project)

https://www.owasp.org/index.php/Main_Page

- OWASP top 10

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

- CWE (Common Weakness Enumeration)

<http://cwe.mitre.org>

- CWE top 25

<http://cwe.mitre.org/top25/index.html>



- SQL Injection is a very real threat to online applications using SQL databases
- SQL Injection is preventable by simply:
 - Limiting user authority
 - Using parameterized APIs
 - Checking/Validating user input



RMUG TRAINING DAYS 2014

February 5-7, 2014 - Denver Convention Center - Denver, Colorado

Tracks

- Application Development
- Business Intelligence
- Database Administration
- DBA Deep Dive
- Database Tools of the Trade
- Hyperion
- Middleware
- Professional Empowerment



PHOTO CREDIT: Mike Landrum, SQL Developer and the "Data Tsunami" from i-Behavior

www.rmoug.org



COLLABORATE 14 – IOUG Forum

April 6 – 10, 2014

The
Venetian
Las Vegas,
NV



ODTUG Kscope14



SEATTLE, WASHINGTON • JUNE 22-26

REGISTER TODAY!

TOPICS:

Application Express | ADF and Fusion Dev. | Developer's Toolkit | The Database
Building Better Software | Business Intelligence | Essbase | Planning | Financial Close
EPM Reporting | EPM Foundations and Data Management | EPM Business Content



www.kscope14.com



Avoiding SQL Injection: Don't Let a Stranger "Shoot you up"

To contact the author:

John King

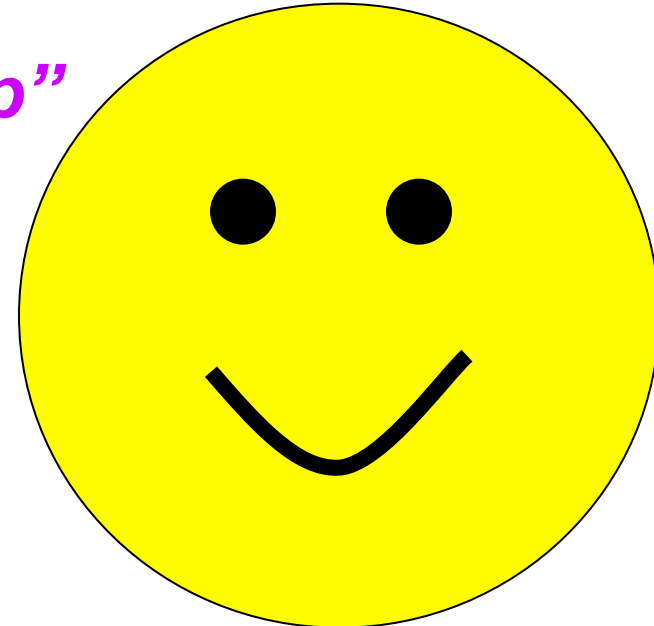
King Training Resources

P. O. Box 1780

Scottsdale, AZ 85252 USA

1.800.252.0652 - 1.303.798.5727

Email: john@kingtraining.com



Thanks for your attention!

Today's slides and examples are on the web:
<http://www.kingtraining.com>