Cheatography

SQL Injection Cheat Sheet

by binca via cheatography.com/44948/cs/13343/

Intro

Perhaps the most well known web app flaw

Easier to address from an app security perspective, but remains a common flaw.

Apps employ relational databases for a multitude of reasons

App interfaces to add, update and render data

Flaw originates from app allowing user-supplied input to be dynamically used in a SQL query

Numerous different Relational Database Management Systems in use including Oracle, MySQL, MSSQL

Key SQL Verbs

SELECT	Retrieves data from tables, most commonly used
INSERT	Add data to table
UPDATE	Modify existing data
DELETE	Delete data in a table
DROP	Delete a table
UNION	Combine data from multiple queries

SQL Query Modifiers

WHERE	Filter SQL query to apply only when a condition is met
*AND/OR**	Combine WHERE to narrow SQL query
LIMIT #1, #2	Limits rows returned to #2, many rows starting at #1, same results with **LIMIT 2 OFFSET 1**
ORDER BY [#]	Sort by column number

Important SQL Data Types

bool	Boolean True/False
int	Integer
char	Fixed length string
varchar	Variable length string
binary	

Note: Names for data types may vary across RDBMSs

SQL Special Chatacters

' , "	String delimiter
;	Terminates a SQL statements
-- , # , /*	Comment delimiters
% , *	Wildcard characters
|| , + , " "	String concatenation characters
+ , < , > , =	Mathematical operators
=	Test for equivalence
( )	Calling functions, sub-queries, and INSERTs
%00	Null byte

SQL Injection Example Code

Server-side PHP code taking the value of URL query parameter name as input to SQL SELECT

```
$ sql="SELECT * FROM Users WHERE lname='$_GET["name"]';"
```

The resulting query if normal input is John
URL: http://url/sqli.php?name=John
SQL Query: SELECT * FROM Users WHERE lname='John';
Normal result.
Injected Input Query
Input is John'
URL: http://url/sqli.php?name=John'
SQL Query: SELECT * FROM Users WHERE lname='John'';
Stray ' causes error.
Inject Input Query 2
Input is John'; --
URL: http://url/sqli.php?name=John';--
SQL Query: SELECT * FROM Users WHERE lname='John';--';
Normal results.

' or 1=1; --

A payload or variation upon that is found in most SQLi documentation

The **single quote*** closes out any string.

The **1=1** changes query logic because it is always true.

;-- Ends the payload completing the statement and comments out the remaining code to prevent syntax errors

Note: Some RDBMS require a space after "--" comment delimiter.

Cheatography

SQL Injection Cheat Sheet

by binca via cheatography.com/44948/cs/13343/

SQLi Balancing Act

Involves finding correct prefixes, payloads and suffixes to evoke desired behavior.

Significant aspect of discovering SQLi flaws is determining reusable pieces of our injection.

Most obvious balancing act is quotes.

The most common data type our input will land within are strings so proper prefixes and suffixes to accommodate strings are necessary.

Example with comments: John';--
SELECT...WHERE lname='John';--';

Example without comments: John' OR '1'='1
SELECT...WHERE lname='John' OR '1'='1';

Balancing Column Numbers and Data Types

INSERT and **UNION** statement require us to know the number of columns required or used, otherwise a DB Syntax Error will occur

INSERT and **UNION** statements also require the data type associated with the columns to be compatible.

ORDER BY [#] is another option where the number is incrementally increased until an error is thrown.

Note: Numbers and strings are typically compatible.

Discovery of SQLi

Input locations that leverage/interact with backend DB such as login functionality.

HTTP Request portions that are common input locations:
GET URL query parameters
POST payload
HTTP COOKIE
HTTP User-agent

HTTP COOKIE and **User-agent** are more likely to be blind.

Classes of SQLi

One vulnerability encountered in a variety of ways

Simplest categorization is blind versus visible, but there is spectrum.

In-Band/Inline SQLi is a flaw that allows us to see the result of our injection. They are easier to discover and exploit.

Blind SQLi is the same vulnerability but with no visible response.

Error Messages

Database Error Messages	Not only hint at the presence of SQLi but may guide us in crafting input for exploitation. If you see **database error messages** it is NOT blind SQLI
Custom Error Messages	Can require a different approach because the error will not indicate if the input is being interpreted.

Equivalent String Injections

Prefix	Suffix	Note
John'	**;#**	Commenting
John'	**;--**	Commenting
Jo'/*	***/'hn**	Inline Commenting
Jo'	**'hn**	Concatenation (with or without spaces)
Jo'|	**|'hn**	Concatenation

Comment delimiters (--, /**/, #) can allow injections to succeed that would otherwise fail.
The -- and # are useful SQL suffixes.
Injecting into the middle of a SQL statement/query will not allow us to alter the rest of the SQL statement but it will show us if our input is being interpreted on the backend when we experience custome error messages (Blind SQLi).

Binary/Boolean Inference Testing

John' AND 1;#	True
John' AND 1=1;#	True
John' AND 0;#	False
John' AND 1=0;#	False

If it evaluates to True (AND 1=1) or False (AND 1=0)
Prefix: Dent' AND
Evaluates: substr((select table_name from information_schema.tables limit 1,),1,1) > "a"
Suffix: ;#

Blind Timing Inferences

When there is no discernible output or errors the use of timing-based inference is a viable option.

Relies on responsiveness of app for the inference by artificially inducing a delay when a condition evaluates.

Example:
Sleep(10) - MySQL
WAITFOR DELAY '0:0:10' - MSSQL

Out-of-Band SQLi

No errors messages

No visible responses

No boolean/inference opportunities without or without timing

Requires an alternative communication channel to discover or exploit these flaws

Out-of-Band Channels may provide for faster ex-filtration of some flaws susceptible to inference techniques. Typically leverages HTTP or DNS to tunnel communications back to attacker controlled server

Query Disclosure

UNION SELECT is used to disclose the vulnerable query we are injecting into.
Payload:
John' UNION SELECT '1','2','3', info FROM information_schema.processlist;#
Results:
SELECT * FROM Customers WHERE lname='John' UNION SELECT '1','2','3'', info FROM information_schema.processlist;#'