Database Fingerprinting Cheat Sheet

by binca via cheatography.com/44948/cs/13351/

Purpose

Determine type of backend DB to guide injection crafting, sometimes an educated guess based on information and configuration reconnaissance or error messages.

Use special function parameters such as **SELECT @@ version** (MySQL and SQL Server)

Unique numeric functions:
MySQL - connnection_id()
MSSQL - @@pack_received
Oracle - BITAND(1,1)

(Meta) Database Information

The RDBMS being used will affect metadata and schema information, which will be used to determine tables, columns, users and passwords

information_schema is an ANSI SQL92 standard database that can provide us with relevant metadata negating the need for fingerprinting, though implementations vary.

MySQL's information_schema includes information for every DB, while **MSSQL** only shows information for the current DB.

Oracle, DB2, and SQLite do **NOT** support information_schema.

information_schema Databases

RDBMS	Databases	Tables	Columns
MySQL	schema_name FROM information_schema.schemata	table_name FROM informaton_schema.tables	column_name FROM information_schema.columns
SQL Server or Azure SQL*	name FROM sys.databases	name FROM sys.tables	name FROM sys.columns
Oracle DB	**...owner FROM all_tables	table_name FROM all_tables	column_name FROM all_tab_columns

*Deprecated syntax master..sysobjects system tables

Exploiting In-Band/In-line SQLi

With a SELECT query we can see all data contained in columns employed, but we are confined to the table the query SELECTs FROM.

To see beyond the current table we can use **Stacked Queries** if they are supported.

Stacked Queries are multiple SQL queries submitted by splitting them with a **;**.

Example: **SELECT * FROM Users WHERE lname='John'; CREATE TABLE exfil(data varchar(1000));--';**

Most often support with **MSSQL**.

MySQL support is complicated because will the DB supports it the way the app interfaces with MySQL limiting the abilities.

Oracle does NOT support Stacked Queries.

Stacked Queries are not required for data retrieval/exfiltration but make it easier.

Stacked Queries are important when we want to do more than SELECT. Enables us to do **INSERTs, UPDATEs, DROPs, SHUTDOWNs** with ease.

Unionizing SQLi

UNION allows us to move beyond the confines of the table so we can access arbitrary data from the DB.

Example: **SELECT * FROM Users WHERE lname='John' UNION SELECT * FROM Customers;--';**

Prerequests:
# of columns being pulled must match in the original and injected SELECT
Column data must be compatible
Know table names to target

FROMless SELECT

SELECT Statements do **NOT** require an associated **FROM**

When the FROM is left out the result is an interpreted form of the supplied input.

SELECT 1; -- returns 1

ORACLE DB requires **FROM** for all **SELECT** statements but provides a built-in **DUAL** table that acts as a dumby

NULL

NULL is compatible with any data type.

Couple with FROMless SELECT with NULL to prevent mismatch of data types.

UNION and NULL

Use **NULL** with **UNION SELECT** to determine number of columns by increasing the number of NULLs until an error is presented.

Example: **SELECT * FROM Users WHERE lname='John' UNION SELECT NULL, NULL, NULL;--';**

This approach also works for **INSERT** statements.

Note: Another method is to determine column numbers with an **ORDER BY** clause.

Data Types

Require at least 1 column that accommodates strings to accept data we exfiltrate

Tweak previous column number injection changing each NULL to a string until the query is successful.

Example: **SELECT * FROM Users WHERE lname='John' UNION SELECT 'string', NULL, NULL;--';**

Data Exfiltration

Using **UNION** and having establish the number of columns and at least one column that accepts strings we can iterate through all columns of interesting tables to return data.

Blind data exfiltration is the same approach as UNION but encumbered by having to use inference techniques.

Tools make this more efficient and in the case of blind data exfiltration make it easier.

SQLi Potential attacks

While data exfiltration is the most commonly performed exploit again SQLi flaws in some cases the data holds little value. Attackers can still perform other attacks.

Deleting or altering valuable data.

Injecting data used as stored XSS payloads

Reading files
MySQL - LOAD_FILE()
SQL Server BULK INSERT

Writing files **MySQL - INTO OUTFILE**

OS interaction beyond files because stored procedures used to interact with the OS may be on the DB

SQLi Shell Access

Writing files can be used to achieve interactive shells (file writing similar to file uploading)

Requirements:
DB server also running web server
DB account needs privileges to write to web root
Have the ability to browse web root

Alternative approaches require **Stacked Queries**.

More viable during internal penetration test or in a pivoted SQLi.

SQLi Cheat Sheets

WebSec SQL Injection Knowledge Base -https://websec.ca/kb/sql_injection

pentestmonkey SQL Injection Cheat Sheet -http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet

SQL Injection Wiki Cheat Sheet -http://www.sqlinjectionwiki.com/

Defensive: OWASP SQL Injection Prevention Cheat Sheet -https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet