

Lab Guide

Hands-on exercises: build and run hashcat commands with different parameters.

Prerequisites: Hashcat installed and working (verified in Lesson 02). A terminal open in the hashcat directory. The `rockyou.txt` wordlist available (or any wordlist of your choice).

Important: All exercises use test hashes that you generate yourself. Never use hashcat against hashes you do not own or have explicit authorization to test. This lab is for educational purposes only.

Setup: Create Test Hashes

Before starting the exercises, generate some known hashes so you can verify that your commands actually work.

Generate MD5 test hashes

Create a file called `test_md5.txt` with MD5 hashes of known passwords:

```
# Linux / macOS
echo -n "password" | md5sum | awk '{print $1}' > test_md5.txt
echo -n "123456" | md5sum | awk '{print $1}' >> test_md5.txt
echo -n "letmein" | md5sum | awk '{print $1}' >> test_md5.txt
echo -n "dragon" | md5sum | awk '{print $1}' >> test_md5.txt
echo -n "sunshine" | md5sum | awk '{print $1}' >> test_md5.txt
```

On macOS, replace `md5sum` with `md5 -r`:

```
echo -n "password" | md5 -r | awk '{print $1}' > test_md5.txt
echo -n "123456" | md5 -r | awk '{print $1}' >> test_md5.txt
echo -n "letmein" | md5 -r | awk '{print $1}' >> test_md5.txt
```

```
echo -n "dragon" | md5 -r | awk '{print $1}' >> test_md5.txt
echo -n "sunshine" | md5 -r | awk '{print $1}' >> test_md5.txt
```

Create a small wordlist

Create `test_wordlist.txt` with a few hundred common passwords (or use `rockyou.txt`). For quick testing, a small list is fine:

```
printf "password\n123456\nletmein\ndragon\nsunshine\nqwerty\nabc123\nmon
```

Exercise 1 — Basic Dictionary Attack

EX 1 Run your first hashcat command with `-m`, `-a`, and `-o`

Step 1: Run a basic MD5 dictionary attack

```
hashcat -m 0 -a 0 -o results_ex1.txt test_md5.txt test_wordlist.txt
```

Observe the output. Hashcat should crack all 5 hashes almost instantly.

Step 2: Inspect the results

```
cat results_ex1.txt
```

You should see lines in the format `hash:password`.

Step 3: Change the output format

Re-run with different `--outfile-format` values and compare:

```
# Password only
hashcat -m 0 -a 0 -o results_plain.txt --outfile-format 2 test_md5.txt test_wordlist.txt

# Hash only
hashcat -m 0 -a 0 -o results_hash.txt --outfile-format 1 test_md5.txt test_wordlist.txt
```

Note: Use `--potfile-disable` to prevent hashcat from skipping already-cracked hashes from the potfile.

Self-check: You should have three output files with different formats. Compare them to confirm you understand what each `--outfile-format` value produces.

Exercise 2 — Mask Attack with Incremental Mode

EX 2 Use `-a 3` with masks and the `-i` flag

Step 1: Create a hash of a short numeric password

```
# Hash the password "4521"
echo -n "4521" | md5sum | awk '{print $1}' > test_mask.txt
```

Step 2: Crack with a fixed-length mask

```
hashcat -m 0 -a 3 -o results_ex2a.txt test_mask.txt ?d?d?d?d
```

This tries all 4-digit combinations (0000–9999). Should crack in under a second.

Step 3: Use incremental mode

Now hash a password of unknown length and use increment:

```
# Hash "83"
echo -n "83" | md5sum | awk '{print $1}' > test_incr.txt

# Crack with increment from 1 to 6 digits
hashcat -m 0 -a 3 -i --increment-min 1 --increment-max 6 \
-o results_ex2b.txt test_incr.txt ?d?d?d?d?d?d
```

Step 4: Observe the keyspace

Watch the hashcat status output. Notice how it processes length 1 first (?d), then length 2 (?d?d), and so on. Press `s` during execution to see status.

Self-check: Both hashes should be cracked. Confirm that the incremental mode found "83" by trying lengths 1 and then 2.

Exercise 3 — Dictionary Attack with Rules

EX 3 Apply rule files with `-r` to expand dictionary coverage

Step 1: Create a hash that a plain dictionary won't crack

```
# "Password1" — capitalized with a digit suffix
echo -n "Password1" | md5sum | awk '{print $1}' > test_rules.txt

# "DRAGON" — all uppercase
echo -n "DRAGON" | md5sum | awk '{print $1}' >> test_rules.txt

# "sunshine123" — word + digits
echo -n "sunshine123" | md5sum | awk '{print $1}' >> test_rules.txt
```

Step 2: Try without rules (will likely fail)

```
hashcat -m 0 -a 0 -o results_ex3a.txt test_rules.txt test_wordlist.txt --
```

Check: did it crack all three? Probably not — the wordlist has "password" but not "Password1".

Step 3: Add the best64 rule file

```
hashcat -m 0 -a 0 -r rules/best64.rule -o results_ex3b.txt \
test_rules.txt test_wordlist.txt --potfile-disable
```

The rules mutate each word (capitalize, append digits, toggle case, etc.). Check if more hashes were cracked.

Step 4: Compare results

```
echo "=== Without rules ==="
cat results_ex3a.txt
echo "=== With rules ==="
cat results_ex3b.txt
```

Self-check: The rules-based attack should crack more (or all) of the hashes. Note which mutations were needed.

Exercise 4 — Hybrid Attack

EX 4Combine dictionary and mask with `-a 6`

Step 1: Create a hash with a word+digits pattern

```
echo -n "monkey99" | md5sum | awk '{print $1}' > test_hybrid.txt
echo -n "master42" | md5sum | awk '{print $1}' >> test_hybrid.txt
```

Step 2: Run a hybrid dictionary + mask attack

```
hashcat -m 0 -a 6 -o results_ex4.txt test_hybrid.txt test_wordlist.txt ?c
```

This appends every 2-digit combination (00–99) to every word in the wordlist.

Step 3: Verify results

```
cat results_ex4.txt
```

Both "monkey99" and "master42" should appear in the output.

Self-check: Both hashes cracked. Think about how this is more efficient than brute-forcing the entire password.

Exercise 5 — Session Management and Workload

EX 5Use `--session`, `--restore`, and `-w`

Step 1: Start a named session with a longer-running task

```
hashcat -m 0 -a 3 --session lab03test -w 1 \
-o results_ex5.txt test_md5.txt ?1?1?1?1?1
```

This tries all 6-character lowercase combinations (~309 million candidates). It may take a few seconds to a minute depending on your hardware.

Step 2: Press `q` to gracefully quit

While the attack is running, press `q` to save progress and exit.

Step 3: Resume the session

```
hashcat --session lab03test --restore
```

Notice that it picks up where it left off, not from the beginning.

Step 4: Experiment with workload profiles

Run the same attack with `-w 3` instead of `-w 1` and compare the speed (hashes/second) shown in the status output.

Self-check: Session restore should continue from where it stopped. Higher workload should show increased speed.

Exercise 6 — Build a Complete Command

EX 6 Combine all parameters into one production-quality command

Challenge

Write a single hashcat command that meets all of these requirements:

- Cracks NTLM hashes (`-m 1000`)
- Uses a mask attack (`-a 3`)
- Tries lengths 4 through 8 (incremental mode)
- Uses all printable characters (`?a`)
- Saves output as hash:password format
- Uses a named session called `ntlm_audit`
- Sets workload to high (`-w 3`)

Expected command

Try to build it yourself before revealing:

```
hashcat -m 1000 -a 3 -o ntlm_results.txt --outfile-format 3 \  
--session ntlm_audit --increment --increment-min 4 --increment-max 8 \  
-w 3 ntlm_hashes.txt ?a?a?a?a?a?a?a
```

Note: You can generate NTLM test hashes using Python:

```
import hashlib  
password = "Test"  
ntlm_hash = hashlib.new('md4', password.encode('utf-16le')).hexdigest()  
print(ntlm_hash)
```

Self-check: Your command should contain all 8 required elements. Count them off one by one.

Clean up: After completing the lab, remove your test files: `rm -f test_*.txt results_*.txt`. Also clear the potfile if desired: `rm -f hashcat.potfile`.