

Lab Guide — Lab 2

Hands-on dictionary attack: download a wordlist, crack known hashes, and explore rule-based mutations.

Prerequisites

- Hashcat installed and working (verified in Lab 1)
- Terminal / command-line access
- Completed the [Theory](#) reading for this lesson
- Internet connection (to download rockyou.txt if needed)

Ethics reminder: Only crack hashes that you have created yourself or have explicit authorization to test. The hashes in this lab are well-known test values.

Step 1 — Download the rockyou.txt Wordlist

Task 1.1 — Obtain rockyou.txt

The rockyou.txt wordlist is the standard starting point for dictionary attacks. Choose one of the following methods:

Option A: Kali Linux (pre-installed)

```
# rockyou.txt ships compressed with Kali
sudo gunzip /usr/share/wordlists/rockyou.txt.gz
ls -lh /usr/share/wordlists/rockyou.txt
# Expected: ~139 MB, ~14.3 million lines
```

Option B: Download from the internet

```
# Download from a common mirror
wget https://github.com/brannondorsey/naive-hashcat/releases/download/data/rockyou.txt
```

```
# Or use curl:
curl -L -o rockyou.txt https://github.com/brannondorsey/naive-hashcat/rel
```

Option C: Create a minimal test wordlist

If you cannot download rockyou.txt, create a small test wordlist that includes the target passwords:

```
cat << 'EOF' > test_wordlist.txt
password
123456
admin
letmein
welcome
monkey
dragon
master
qwerty
login
EOF
```

Checkpoint: Verify your wordlist exists and has content.

```
wc -l rockyou.txt
# Expected: 14344391 (or 10 lines if using the mini test list)
```

Step 2 — Create the Target Hash File

Task 2.1 — Create hashes.txt

Create a file containing two MD5 hashes. These are the MD5 hashes of commonly known passwords:

```
cat << 'EOF' > hashes.txt
5f4dcc3b5aa765d61d8327deb882cf99
e10adc3949ba59abbe56e057f20f883e
EOF
```

For reference, these hashes correspond to:

Hash	Expected Password
5f4dcc3b5aa765d61d8327deb882cf99	password
e10adc3949ba59abbe56e057f20f883e	123456

Checkpoint: Verify the hash file.

```
cat hashes.txt
# Should show exactly 2 lines, each a 32-character hex string
```

Step 3 — Run the Dictionary Attack

Task 3.1 — Execute the attack

Run hashcat in dictionary mode against your hashes:

```
hashcat -m 0 -a 0 -o results.txt hashes.txt rockyou.txt
```

Parameter review:

- `-m 0` — MD5 hash type
- `-a 0` — Dictionary (straight) attack mode
- `-o results.txt` — Write cracked pairs to this file
- `hashes.txt` — Input hashes
- `rockyou.txt` — Wordlist

If you get a message that the hashes have already been cracked (from hashcat's potfile cache), use:

```
# Force re-crack by ignoring the potfile
hashcat -m 0 -a 0 -o results.txt --potfile-disable hashes.txt rockyou.txt
```

Checkpoint: The attack should complete in seconds. Both hashes should be cracked.

Task 3.2 — Examine the results

```
cat results.txt
```

Expected output:

```
5f4dcc3b5aa765d61d8327deb882cf99:password  
e10adc3949ba59abbe56e057f20f883e:123456
```

The format is `hash:password`. Both passwords are among the most common in any breach dataset, so they are found almost instantly.

Task 3.3 — Review hashcat status output

While hashcat runs (or after, using `--show`), note these key metrics:

- **Speed:** How many hashes per second (H/s) is your system achieving?
- **Progress:** How many words from the wordlist were tested?
- **Recovered:** How many of the 2 hashes were cracked?

```
# Show previously cracked hashes from the potfile  
hashcat -m 0 hashes.txt --show
```

Step 4 — Run with Rules

Task 4.1 — Add best64.rule

Re-run the attack with the `best64.rule` file to see the effect of rule-based mutations:

```
# Clear the potfile first so hashes are re-cracked
hashcat -m 0 -a 0 -r rules/best64.rule -o results_rules.txt --potfile-dis
```

Observe the difference in candidate count and speed compared to the plain dictionary attack.

Where are the rules? Rule files are in hashcat's installation directory under `rules/`. On Kali Linux: `/usr/share/hashcat/rules/`. On macOS with Homebrew: `/opt/homebrew/share/hashcat/rules/` or `/usr/local/share/hashcat/rules/`. You can also find the path with: `find / -name "best64.rule" 2>/dev/null`.

Task 4.2 — Compare performance

Fill in this comparison based on your observations:

Metric	Plain Dictionary	Dictionary + best64.rule
Total candidates	~14.3M	~14.3M x 64 = ~916M
Time to crack both hashes	(your value)	(your value)
Hash speed (H/s)	(your value)	(your value)

For these trivial hashes, both approaches crack them instantly. The real advantage of rules appears with more complex passwords like "Password1!" or "p@ssw0rd".

Step 5 (Bonus) — Combinator Attack

Task 5.1 — Create test dictionaries

```
# Create two small dictionaries
cat << 'EOF' > colors.txt
red
blue
green
dark
light
EOF

cat << 'EOF' > things.txt
sky
dragon
fire
wolf
star
EOF
```

Task 5.2 — Generate a test hash

```
# Generate MD5 hash of "bluedragon"
echo -n "bluedragon" | md5sum
# Expected: 53153a856798e842d84e5be22a4b8e37
```

```
# On macOS, use md5 instead:
echo -n "bluedragon" | md5
# Or: md5 -s "bluedragon"
```

```
# Save the hash
echo "53153a856798e842d84e5be22a4b8e37" > combo_hash.txt
```

Task 5.3 — Run the combinator attack

```
hashcat -m 0 -a 1 -o combo_results.txt --potfile-disable combo_hash.txt c
cat combo_results.txt
# Expected: 53153a856798e842d84e5be22a4b8e37:bluedragon
```

Checkpoint: The combinator attack should find "bluedragon" by joining "blue" from the first dictionary with "dragon" from the second.

Completion Checklist

- rockyou.txt wordlist obtained and verified
- hashes.txt created with 2 MD5 hashes
- Dictionary attack executed successfully; found "password" and "123456"
- results.txt contains both cracked hash:password pairs
- Rule-based attack executed with best64.rule
- Performance comparison table filled in
- (Bonus) Combinator attack cracked "bluedragon"