

Lab Guide — Results Analysis & Post-Processing

Four exercises covering output formats, session management, data cleaning, and password analysis with Python.

Prerequisites: You need hashcat installed and working (from previous lessons), Python 3.8+, and the `pandas` and `matplotlib` packages. Install them with: `pip install pandas matplotlib`

Setup — Create Test Hashes

Before starting, generate a set of test hashes to work with throughout the lab.

Prepare test data

Create a file called `lab_hashes.txt` with MD5 hashes. You can generate these from known passwords:

```
# Generate MD5 hashes from a list of known passwords
echo -n "password" | md5sum | cut -d' ' -f1 >> lab_hashes.txt
echo -n "123456" | md5sum | cut -d' ' -f1 >> lab_hashes.txt
echo -n "qwerty" | md5sum | cut -d' ' -f1 >> lab_hashes.txt
echo -n "letmein" | md5sum | cut -d' ' -f1 >> lab_hashes.txt
echo -n "admin" | md5sum | cut -d' ' -f1 >> lab_hashes.txt
echo -n "welcome1" | md5sum | cut -d' ' -f1 >> lab_hashes.txt
echo -n "monkey" | md5sum | cut -d' ' -f1 >> lab_hashes.txt
echo -n "dragon" | md5sum | cut -d' ' -f1 >> lab_hashes.txt
echo -n "master" | md5sum | cut -d' ' -f1 >> lab_hashes.txt
echo -n "P@ssw0rd!" | md5sum | cut -d' ' -f1 >> lab_hashes.txt

# Also create a version with usernames
echo "alice:$(echo -n 'password' | md5sum | cut -d' ' -f1)" >> lab_users.txt
echo "bob:$(echo -n '123456' | md5sum | cut -d' ' -f1)" >> lab_users.txt
```

```
echo "charlie:$(echo -n 'qwerty' | md5sum | cut -d' ' -f1)" >> lab_users.txt
echo "diana:$(echo -n 'letmein' | md5sum | cut -d' ' -f1)" >> lab_users.txt
echo "admin:$(echo -n 'admin' | md5sum | cut -d' ' -f1)" >> lab_users.txt
```

Exercise 1 — Output Formats

EX 1

Explore --outfile-format options

Run the same attack with different output formats and compare the results.

1. Run a dictionary attack saving results in format 3 (default):

```
hashcat -m 0 lab_hashes.txt /usr/share/wordlists/rockyou.txt \
-o results_fmt3.txt --outfile-format 3
```

2. Run again with format 2 (plain passwords only):

```
hashcat -m 0 lab_hashes.txt /usr/share/wordlists/rockyou.txt \
-o results_fmt2.txt --outfile-format 2 --potfile-disable
```

3. Run with format 5 using the username file:

```
hashcat -m 0 lab_users.txt /usr/share/wordlists/rockyou.txt \
-o results_fmt5.txt --outfile-format 5 --username
```

4. Compare the three output files:

```
echo "=== Format 3 (hash:password) ==="
head -5 results_fmt3.txt
echo "=== Format 2 (plain only) ==="
head -5 results_fmt2.txt
echo "=== Format 5 (login:password:hash) ==="
head -5 results_fmt5.txt
```

Checkpoint: You have three output files, each showing the same cracked passwords in different formats.

Exercise 2 — Session Management

EX 2

Practice --session and --restore

1. Start a named session with a large wordlist:

```
hashcat -m 0 lab_hashes.txt /usr/share/wordlists/rockyou.txt \  
--session lab09_session -o session_results.txt
```

2. While running, press `s` to view status, then press `p` to pause
3. Press `q` or `Ctrl+C` to quit gracefully (saves checkpoint)
4. Resume the session:

```
hashcat --session lab09_session --restore
```

5. After completion, inspect what was cracked and what remains:

```
# Show cracked hashes  
hashcat -m 0 --show lab_hashes.txt  
  
# Show uncracked hashes  
hashcat -m 0 --left lab_hashes.txt  
  
# Save uncracked to a separate file  
hashcat -m 0 --left lab_hashes.txt > remaining.txt  
echo "Remaining hashes:"  
wc -l remaining.txt
```

Checkpoint: You can pause, resume, and inspect results from a named session.

Exercise 3 — Python Data Cleaning (clean_results.py)

EX 3

Build a script to clean and normalize hashcat output

Create a file called `clean_results.py` that processes raw hashcat output.

```
#!/usr/bin/env python3
"""clean_results.py - Clean and normalize hashcat output."""
import pandas as pd
import sys

def clean_results(input_file, output_file='clean_results.csv'):
    # Read hash:password format
    data = []
    with open(input_file, 'r') as f:
        for line in f:
            line = line.strip()
            if ':' in line:
                # Split on first colon only (passwords may contain colons)
                parts = line.split(':', 1)
                if len(parts) == 2:
                    data.append({'hash': parts[0], 'password': parts[1]})

    df = pd.DataFrame(data)
    print(f"Total entries read:      {len(df)}")

    # Remove duplicate hashes (same hash cracked multiple times)
    df.drop_duplicates(subset='hash', inplace=True)
    print(f"After dedup by hash:      {len(df)}")

    # Remove empty passwords
    df = df[df['password'].str.len() > 0]
```

Run the script:

```
python3 clean_results.py results_fmt3.txt
```

Checkpoint: The script reads raw hashcat output, removes duplicates, and produces a clean CSV with metadata columns.

Exercise 4 — Password Analysis (analyze_passwords.py)

Build an analysis script with frequency charts and complexity breakdown

Create `analyze_passwords.py` that reads the cleaned CSV and produces visualizations.

```
#!/usr/bin/env python3
"""analyze_passwords.py - Password frequency analysis with charts."""
import pandas as pd
import matplotlib.pyplot as plt
import re
import sys

def classify_complexity(pw):
    """Classify password into Weak/Fair/Good/Strong."""
    checks = [
        bool(re.search(r'[a-z]', str(pw))),
        bool(re.search(r'[A-Z]', str(pw))),
        bool(re.search(r'[0-9]', str(pw))),
        bool(re.search(r'^A-Za-z0-9', str(pw))),
    ]
    score = sum(checks)
    if score <= 1: return 'Weak'
    if score == 2: return 'Fair'
    if score == 3: return 'Good'
    return 'Strong'

def analyze(csv_file):
    df = pd.read_csv(csv_file)
    print(f"Analyzing {len(df)} passwords...\n")

    # --- Top 10 passwords ---
```

Run the analysis:

```
python3 analyze_passwords.py clean_results.csv
```

Checkpoint: You have a bar chart of the top passwords, a length distribution histogram, a complexity pie/bar chart, and printed summary statistics.

Tip: If you are working in a headless environment (no display), add

`plt.switch_backend('Agg')` at the top of the script so matplotlib saves files without trying to open a window.