

Results Analysis & Post-Processing

Cracking hashes is only half the job. This lesson covers how to save, organize, analyze, and report on cracked passwords using hashcat's built-in tools and Python post-processing.

Learning Objectives

- Configure **output file formats** with `--outfile-format`
 - Save and manage results with `-o` and **session management**
 - Use `--show` and `--left` to inspect cracked and uncracked hashes
 - Build **Python scripts** for cleaning, analyzing, and visualizing password data
 - Create professional **password analysis reports**
-

Output File Formats

The `--outfile-format` flag controls the structure of saved results. Choose the format that matches your reporting needs.

Value	Format	Example Output
1	Hash only	5f4dcc3b5aa765d61d8327deb882cf99
2	Plain only	password
3	Hash:Password	5f4dcc3b...:password
4	Login:Password	admin:password
5	Login:Password:Hash	admin:password:5f4dcc3b...
6	Login:Password:Hash:Salt	admin:password:5f4dcc3b...:salt123

Default is format 3 (hash:password). Use format 2 for feeding into other tools. Use format 5 or 6 for comprehensive audit reports.

Saving Results & Session Management

Hashcat provides robust mechanisms for saving output and resuming interrupted sessions.

Saving Results

```
# Save cracked passwords to a file
hashcat -m 0 hashes.txt wordlist.txt -o results.txt

# Save with specific format (login:password)
hashcat -m 0 hashes.txt wordlist.txt -o results.txt --outfile-format 4
```

Session Management

```
# Start a named session
hashcat -m 0 hashes.txt wordlist.txt --session mysession

# Resume an interrupted session
hashcat --session mysession --restore
```

Sessions save state automatically. If a long-running attack is interrupted (power loss, Ctrl+C), use `--restore` to pick up exactly where you left off.

Inspecting Results: --show and --left

After cracking, use these flags to review what was and was not recovered.

--show (Cracked)

Displays all previously cracked hashes from the potfile.

```
hashcat -m 0 --show hashes.txt
```

Output: `hash:password` pairs for every cracked entry.

--left (Uncracked)

Shows hashes that have not been cracked yet.

```
hashcat -m 0 --left hashes.txt
```

Output: remaining hashes that need further attack strategies.

Tip: Pipe `--left` output to a new file and use it as input for more aggressive attacks (larger wordlists, rule stacking, brute-force).

Python Post-Processing

Raw hashcat output needs cleaning and analysis. Python with `pandas` and `matplotlib` is ideal for this.

Cleaning Results

```
import pandas as pd

# Read hashcat output (format 3: hash:password)
df = pd.read_csv('results.txt', sep=':', header=None,
                 names=['hash', 'password'])
df.drop_duplicates(inplace=True)
df.to_csv('clean_results.csv', index=False)
print(f"Unique passwords: {df['password'].nunique()}")
```

Frequency Analysis

```
# Top 10 most common passwords
top = df['password'].value_counts().head(10)
print(top)
```

Why automate? Real-world audits may crack thousands of passwords. Manual review is impractical — scripts make analysis repeatable and auditable.

Visualization & Reporting

Transform raw data into actionable insights with charts and summary statistics.

```
import matplotlib.pyplot as plt

# Password length distribution
df['length'] = df['password'].str.len()
df['length'].hist(bins=20, color='#1f6feb', edgecolor='#0d1117')
plt.xlabel('Password Length')
plt.ylabel('Count')
plt.title('Password Length Distribution')
plt.savefig('length_distribution.png', dpi=150, bbox_inches='tight')
plt.show()
```

Key Report Metrics

- **Crack rate:** % of hashes recovered vs total
- **Top passwords:** Most common passwords (often "password", "123456")
- **Length distribution:** How many passwords are under 8 characters

- **Complexity analysis:** % with uppercase, digits, special characters
-

Complexity Analysis

Categorize cracked passwords by their complexity to identify policy weaknesses.

```
import re

def classify(pw):
    has_upper = bool(re.search(r'[A-Z]', pw))
    has_lower = bool(re.search(r'[a-z]', pw))
    has_digit = bool(re.search(r'[0-9]', pw))
    has_special = bool(re.search(r'[!@#$%^&*~.-_]', pw))
    score = sum([has_upper, has_lower, has_digit, has_special])
    if score <= 1: return 'Weak'
    if score == 2: return 'Fair'
    if score == 3: return 'Good'
    return 'Strong'

df['complexity'] = df['password'].apply(classify)
print(df['complexity'].value_counts())
```

Common finding: In most audits, 60-80% of cracked passwords fall into the "Weak" or "Fair" categories, revealing inadequate password policies.

Lab & Homework

Lab Exercises

1. **Output formats** — run attacks with different `--outfile-format` values
2. **Session management** — use `--session` and `--restore`
3. **Python cleaning** — build `clean_results.py`
4. **Analysis** — build `analyze_passwords.py` with charts

Homework Tasks

1. Full password analysis report with visualizations
2. Complexity breakdown and policy recommendations
3. Automated report generation script

