

Homework

Three tasks building on the lab: a full analysis report, complexity policy evaluation, and an automated report generator.

Complete all four lab exercises before starting the homework. These tasks extend the analysis skills from the lab into professional audit deliverables.

Task 1 — Full Password Analysis Report

Objective

Produce a comprehensive password analysis report from a larger dataset with at least 50 cracked passwords.

Instructions

1. Create a hash file with at least 50 MD5 hashes (use common passwords from a wordlist to generate them, or use the hashes from previous lab exercises)
2. Run a dictionary attack with `-o results.txt --outfile-format 3`
3. Use your `clean_results.py` to produce a clean CSV
4. Use your `analyze_passwords.py` to generate:
 - Top 10 most common passwords (bar chart)
 - Password length distribution (histogram)
 - Complexity breakdown (bar chart)
5. Write a text report (plain text or HTML) that includes:
 - Executive summary: crack rate, key findings
 - Methodology: hash type, attack mode, wordlist used
 - Analysis section with the three charts
 - At least 3 specific recommendations for improving password policy

Deliverable

A report file (`report.txt` or `report.html`) plus the three chart image files.

Task 2 — Complexity Policy Evaluation

Objective

Evaluate cracked passwords against common organizational password policies and measure compliance.

Instructions

1. Define three password policies to test against:

```
policies = {  
    'Basic':    {'min_length': 6, 'require_upper': False,  
                 'require_digit': False, 'require_special': False},  
    'Standard': {'min_length': 8, 'require_upper': True,  
                 'require_digit': True, 'require_special': False},  
    'Strong':   {'min_length': 12, 'require_upper': True,  
                 'require_digit': True, 'require_special': True},  
}
```

2. Write a function `check_policy(password, policy)` that returns `True` if the password meets the policy requirements
3. For each policy, calculate what percentage of cracked passwords would have been compliant
4. Create a grouped bar chart showing compliance rates across all three policies
5. Answer: Which policy would have prevented the most cracks? What is the trade-off with user experience?

Deliverable

A Python script (`policy_check.py`) that outputs compliance percentages and a comparison chart.

Task 3 — Automated Report Generator

Objective

Build a single script that takes a raw hashcat output file and produces a complete analysis report automatically.

Instructions

1. Create `generate_report.py` that accepts command-line arguments:

```
python3 generate_report.py results.txt --total-hashes 100 \
--output report.html
```

2. The script should:
 - Read and clean the results (remove duplicates, handle colons in passwords)
 - Generate all three charts (top passwords, length distribution, complexity)
 - Calculate summary statistics (crack rate, average length, policy compliance)
 - Output an HTML report with embedded charts (use base64 encoding for images)
3. The generated HTML report should be self-contained (no external dependencies) and presentable to a client
4. Include error handling: what if the input file is empty? What if it uses an unexpected format?

Starter template

```
#!/usr/bin/env python3
"""generate_report.py - Automated password audit report generator."""
import argparse
import base64
import io
import pandas as pd
import matplotlib
matplotlib.use('Agg') # Non-interactive backend
import matplotlib.pyplot as plt
```

```
def parse_args():
    parser = argparse.ArgumentParser(
        description='Generate password audit report from hashcat output')
    parser.add_argument('input_file', help='Hashcat output file (format 3)')
    parser.add_argument('--total-hashes', type=int, required=True,
                        help='Total number of hashes in the original file')
    parser.add_argument('--output', default='report.html',
                        help='Output report filename')
    return parser.parse_args()

def fig_to_base64(fig):
    """Convert matplotlib figure to base64 string for HTML embedding."""
    buf = io.BytesIO()
    fig.savefig(buf, format='png', dpi=150, bbox_inches='tight')
    buf.seek(0)
    return base64.b64encode(buf.read()).decode('utf-8')
```

Deliverable

The script `generate_report.py` plus a sample generated `report.html` file.

Tip: Use the `fig_to_base64()` helper to embed charts directly in the HTML as `` so the report is a single self-contained file.

Submission Checklist

- Task 1: Full analysis report with 3 charts, executive summary, and recommendations
- Task 2: Policy compliance script with comparison chart and written analysis
- Task 3: Automated report generator script and sample HTML output
- All Python scripts run without errors

- No raw passwords or identifying information in submitted reports (use aggregate statistics only)