🔐 Password Cracking & Credential Attack Suite (Ethical Security Assessment Project)

Ayyavari Vamshi Krishna

1. Introduction

Passwords remain one of the most commonly used authentication mechanisms in modern computing systems. However, weak password practices such as short length, predictable patterns, and dictionary-based passwords make systems vulnerable to credential-based attacks.

This project, **Password Cracking & Credential Attack Suite**, is designed to demonstrate how attackers exploit weak passwords using dictionary and brute-force techniques, and how defenders can analyze and strengthen password security. The project is implemented strictly in a **controlled and ethical environment** for educational purposes.

2. Project Objectives

The main objectives of this project are:

- To understand common password attack techniques.
- To simulate dictionary-based and brute-force password attacks.
- To analyze password strength using complexity and entropy.
- To generate a security audit report with mitigation recommendations.
- To demonstrate both **Red Team (attack)** and **Blue Team (defense)** perspectives.

3. Ethical Considerations

⚠️ Important Note

This project is developed strictly for:

- Educational purposes
- Authorized security testing
- Controlled laboratory environments

No real user accounts, live systems, or unauthorized credentials were targeted.
Unauthorized password cracking is illegal and unethical.

4. System Requirements & Tools Used

4.1 Operating System

Kali Linux

4.2 Programming Language

Python 3

4.3 Development Tools

Kali Linux Terminal

`nano` text editor

4.4 Python Libraries Used

`itertools`

`string`

`math`

`datetime`

(All libraries are part of Python's standard library.)

5. Project Architecture & Workflow

Workflow Steps:

Dictionary generation using common patterns.

Brute-force password attack simulation.

Password strength analysis (entropy & complexity).

Final security audit report generation.

Logical Flow:

Input → Dictionary Generation → Brute Force Simulation
→ Password Strength Analysis → Audit Report

6. Project Structure

```
Password-Attack-Suite/
 |
├── dictionary_generator.py
├── brute_force.py
├── password_analyzer.py
├── report_generator.py
├── generated_wordlist.txt
├── security_audit_report.txt
```

7. Module 1: Dictionary Generator

Purpose

To generate realistic password wordlists using common patterns attackers rely on.

Techniques Used

- Name-based passwords
- Year appending
- Capitalization
- Leet-speak substitutions

Code: `dictionary_generator.py`

```
names = ["admin", "user", "test"]
years = ["1999", "2000", "2023", "2024"]
symbols = ["!", "@", "#"]

leet_map = {"a":"@", "e":"3", "i":"1", "o":"0", "s":"$"}

def leet_transform(word):
    for char, rep in leet_map.items():
        word = word.replace(char, rep)
    return word

def generate_dictionary():
    wordlist = set()
    for name in names:
        wordlist.add(name)
        wordlist.add(name.capitalize())
        for year in years:
            wordlist.add(name + year)
            wordlist.add(name.capitalize() + year)
        for sym in symbols:
            wordlist.add(name + sym)
        wordlist.add(leet_transform(name))
    return wordlist
```

Output

Generated password list stored in `generated_wordlist.txt`.

8. Module 2: Brute-Force Simulator

Purpose

To simulate brute-force password attacks and demonstrate how password length affects security.

Features

- Incremental password guessing
- Attempt counter
- Time calculation

Code: `brute_force.py`

```
import itertools
import string
import time

TARGET_PASSWORD = "ab1"
CHARSET = string.ascii_lowercase + string.digits

def brute_force():
    attempts = 0
    start_time = time.time()
    for length in range(1, 6):
        for guess in itertools.product(CHARSET, repeat=length):
            attempts += 1
            if ''.join(guess) == TARGET_PASSWORD:
                print("Password Found:", ''.join(guess))
                print("Attempts:", attempts)
                print("Time Taken:", time.time() - start_time)
                return
```

Result

- Demonstrates how weak passwords are cracked quickly.
- Shows exponential growth in attempts as password length increases.

9. Module 3: Password Strength Analyzer

Purpose

To evaluate password security from a defensive (Blue Team) perspective.

Analysis Factors

- Length
- Uppercase & lowercase characters
- Numbers & symbols
- Entropy calculation

Code: `password_analyzer.py`

```
import math
import string

def calculate_entropy(password):
    charset = 0
    if any(c.islower() for c in password): charset += 26
    if any(c.isupper() for c in password): charset += 26
    if any(c.isdigit() for c in password): charset += 10
    if any(c in string.punctuation for c in password):
        charset += len(string.punctuation)
    return len(password) * math.log2(charset)

def analyze_password(password):
    score = 0
    if len(password) >= 8: score += 1
    if any(c.islower() for c in password): score += 1
    if any(c.isupper() for c in password): score += 1
    if any(c.isdigit() for c in password): score += 1
    if any(c in string.punctuation for c in password): score += 1
    entropy = calculate_entropy(password)
    return score, entropy
```

Output

- Password classified as **Weak / Medium / Strong / Very Strong**
- Entropy value displayed

10. Module 4: Security Audit Report Generator

Purpose

To generate a final audit report summarizing vulnerabilities and recommendations.

Code: `report_generator.py`

```
from datetime import datetime

def generate_report():
    report = []
    report.append("PASSWORD SECURITY AUDIT REPORT")
    report.append(f"Date: {datetime.now()}")
    report.append("Weak passwords are vulnerable to brute-force attacks.")
    report.append("Recommendations:")
    report.append("- Use long, complex passwords")
    report.append("- Avoid predictable patterns")
    return "\n".join(report)
```

Output

`security_audit_report.txt` containing findings and mitigation steps.

11. Results & Analysis

- Weak passwords were cracked quickly using brute-force methods.
- Short and predictable passwords had low entropy.
- Strong passwords significantly increased attack complexity.
- Entropy directly correlates with password strength.

12. Learning Outcomes

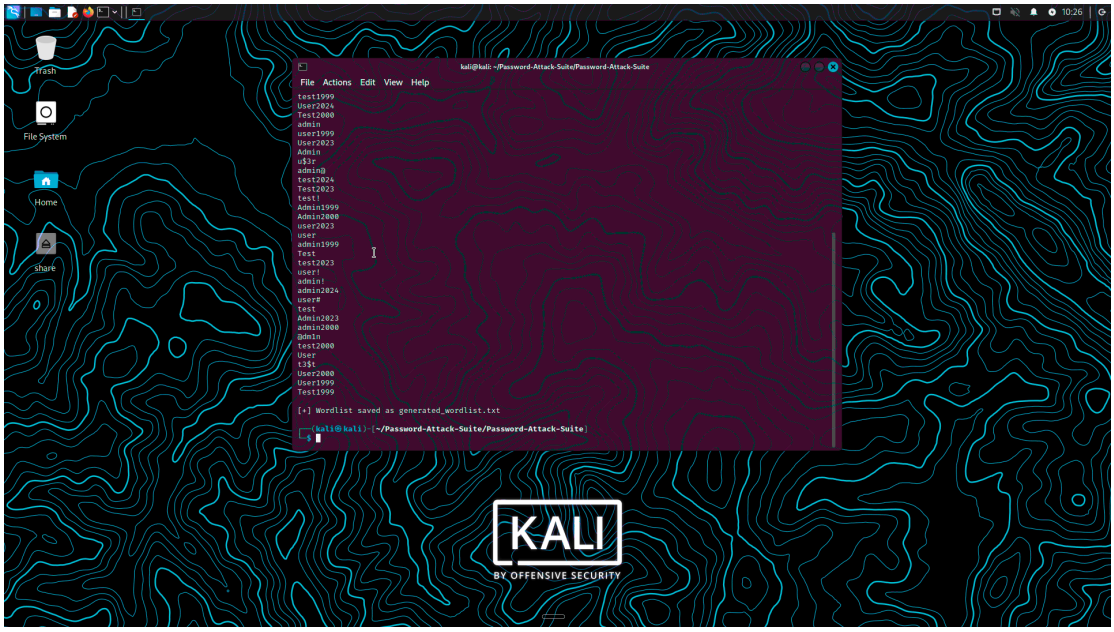
This project helped in understanding:

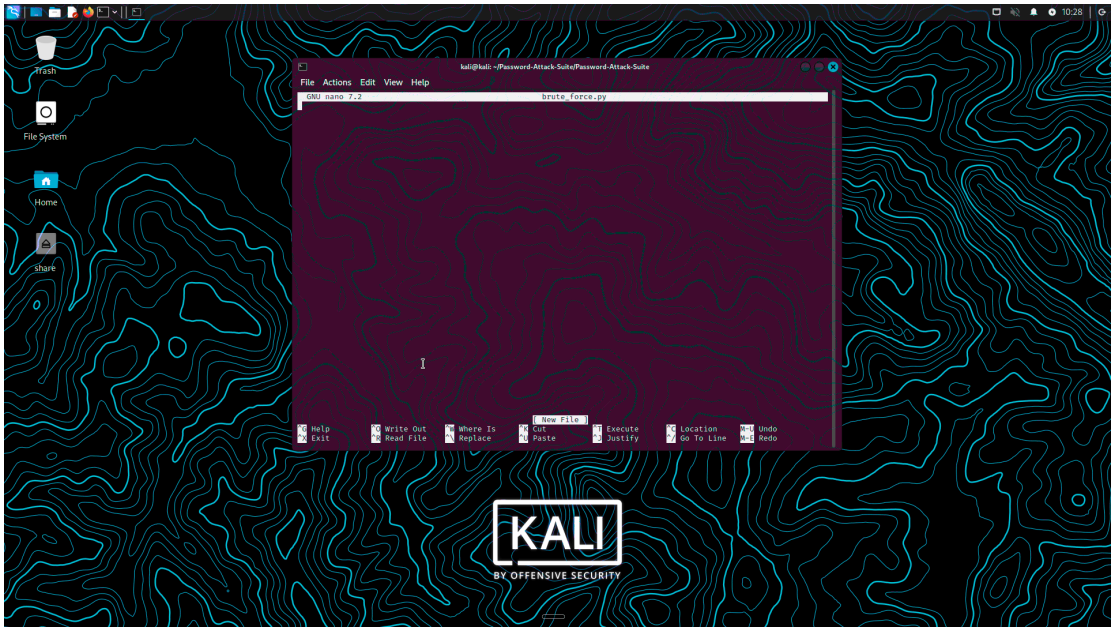
- How password attacks are performed.
- Why weak passwords are dangerous.
- How to analyze password strength.
- Red Team vs Blue Team security approaches.
- Ethical security testing practices.

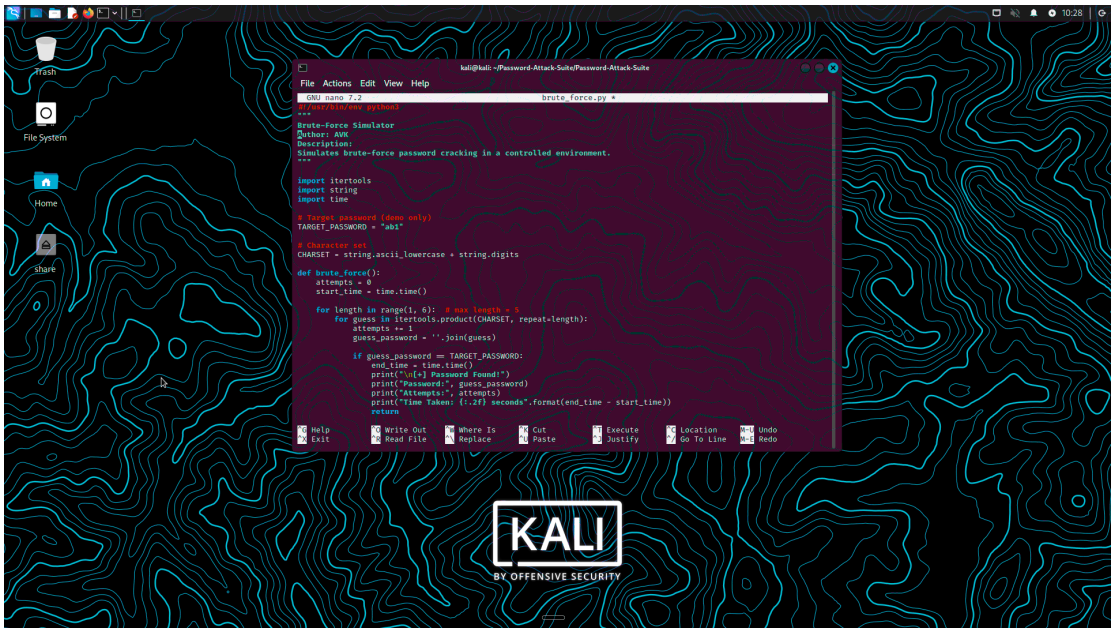
13. Conclusion

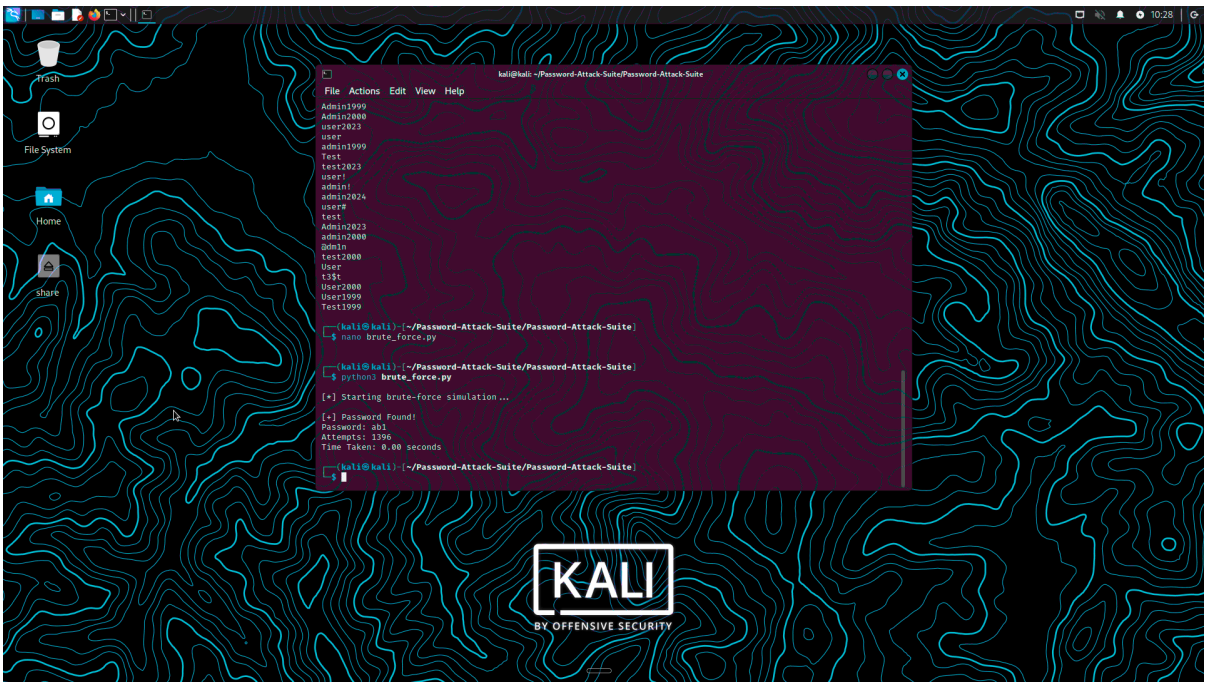
This project successfully demonstrated the risks associated with weak password practices through dictionary-based and brute-force simulations. By integrating password strength analysis and audit reporting, the project highlights the importance of strong password policies in securing systems against credential-based attacks.

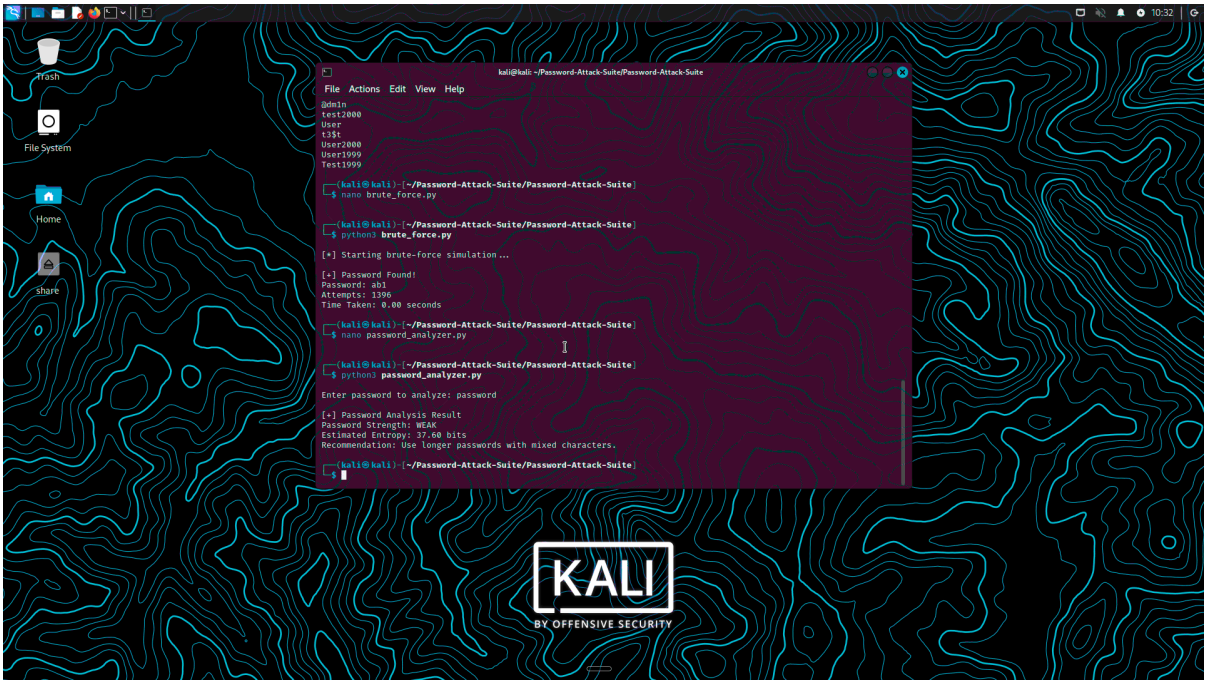
Screenshots

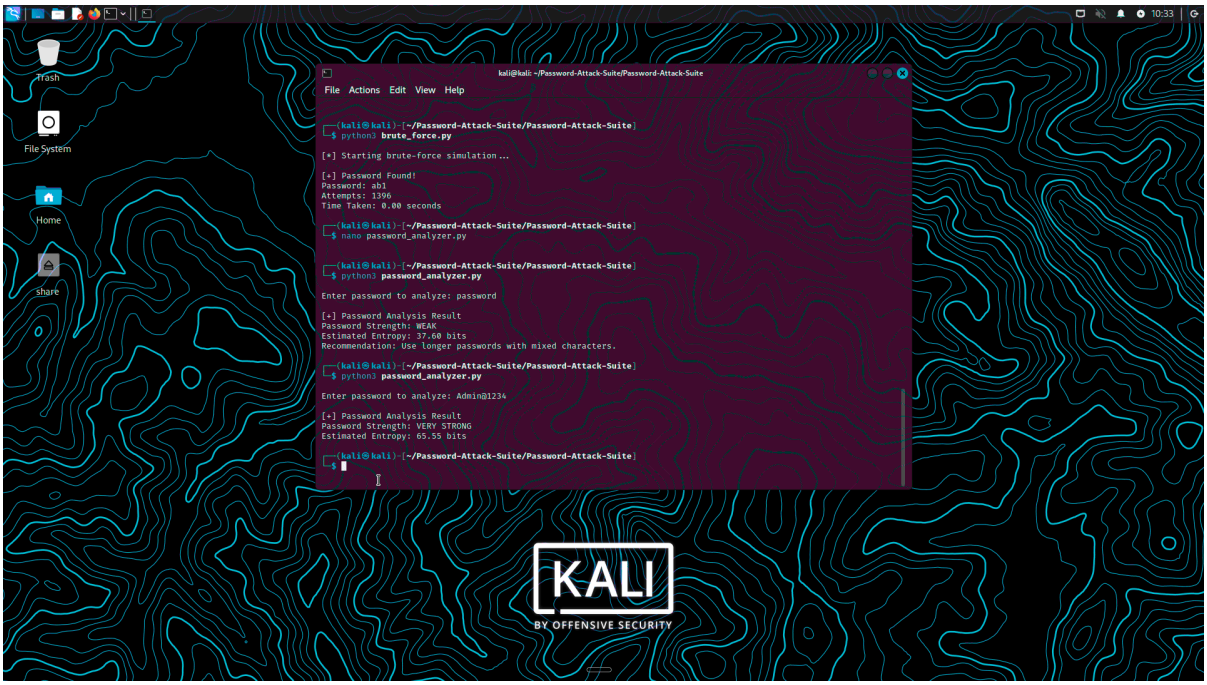

kali@kali: ~/Password-Attack-Suite/Password-Attack-Suite
File Actions Edit View Help
test1999
User2024
Test2000
admin
user1999
User2023
Admin
u$3r
admin@
test2024
Test2023
test!
Admin1999
Admin2000
user2023
user
admin1999
Test
test2023
user!
admin!
admin2024
user#
test
Admin2023
admin2000
@dmin
test2000
User
t3$t
User2000
User1999
Test1999
[+] Wordlist saved as generated_wordlist.txt
(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
KALI
BY OFFENSIVE SECURITY

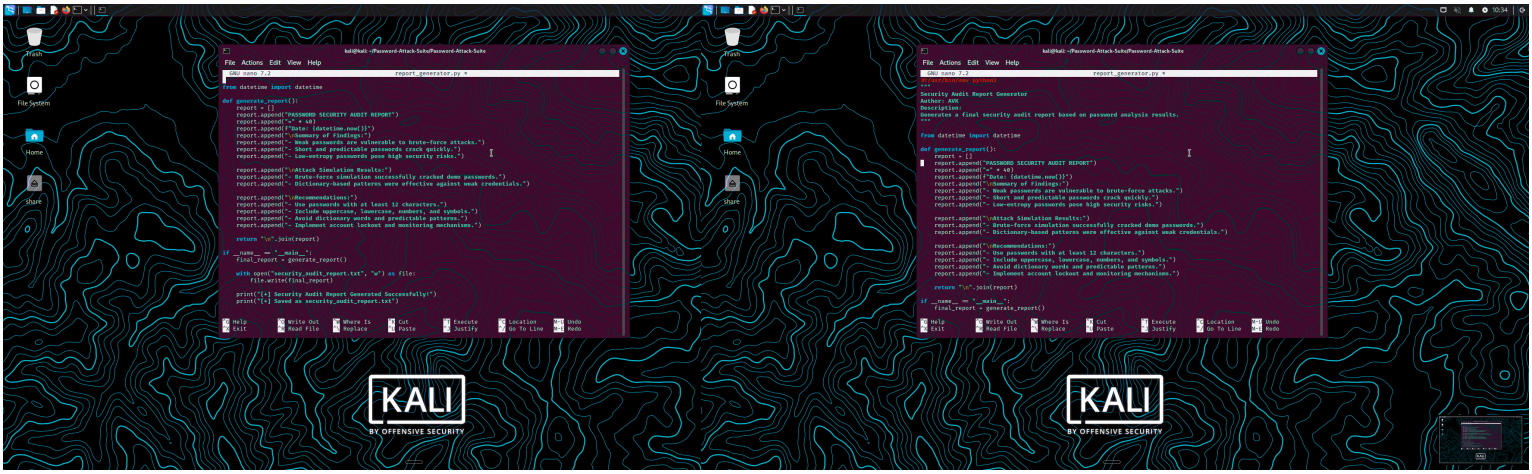

kali@kali: ~/Password-Attack-Suite/Password-Attack-Suite
File Actions Edit View Help
GNU nano 7.2
brute_force.py
[New File]
^G Help
^X Exit
^O Write Out
^R Read File
^W Where Is
^\ Replace
^K Cut
^U Paste
^T Execute
^J Justify
^C Location
M-G Go To Line
M-U Undo
M-E Redo
KALI
BY OFFENSIVE SECURITY

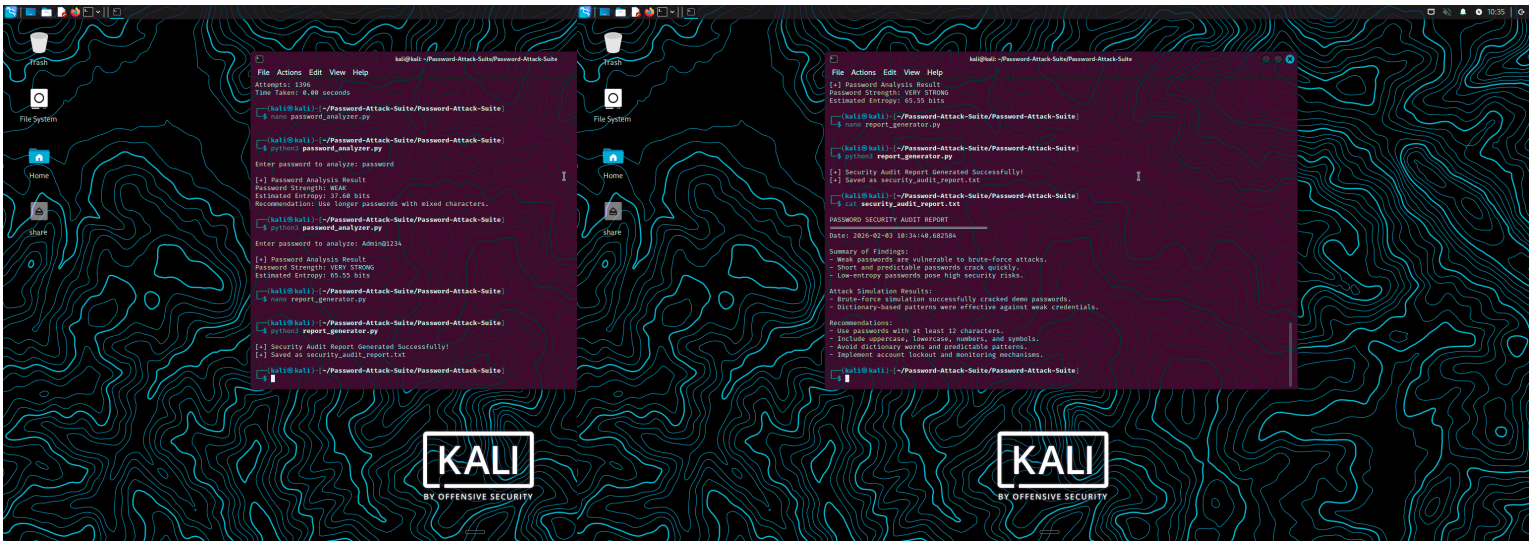

kali@kali: ~/Password-Attack-Suite/Password-Attack-Suite
File Actions Edit View Help
GNU nano 7.2
brute_force.py *
#!/usr/bin/env python3
"""
Brute-Force Simulator
Author: AVK
Description:
Simulates brute-force password cracking in a controlled environment.
"""
import itertools
import string
import time
# Target password (demo only)
TARGET_PASSWORD = "ab1"
# Character set
CHARSET = string.ascii_lowercase + string.digits
def brute_force():
attempts = 0
start_time = time.time()
for length in range(1, 6): # max length = 5
for guess in itertools.product(CHARSET, repeat=length):
attempts += 1
guess_password = ''.join(guess)
if guess_password == TARGET_PASSWORD:
end_time = time.time()
print("\n[+] Password Found!")
print("Password:", guess_password)
print("Attempts:", attempts)
print("Time Taken: {:.2f} seconds".format(end_time - start_time))
return
^G Help
^X Exit
^O Write Out
^R Read File
^W Where Is
^\ Replace
^K Cut
^U Paste
^T Execute
^J Justify
^C Location
M-G Go To Line
M-U Undo
M-E Redo
KALI
BY OFFENSIVE SECURITY


Trash
File System
Home
share
kali@kali: ~/Password-Attack-Suite/Password-Attack-Suite
File Actions Edit View Help
Admin1999
Admin2000
user2023
user
admin1999
Test
test2023
user!
admin!
admin2024
user#
test
Admin2023
admin2000
@dmin
test2000
User
t3$t
User2000
User1999
Test1999
┌──(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
└─$ nano brute_force.py
┌──(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
└─$ python3 brute_force.py
[*] Starting brute-force simulation...
[+] Password Found!
Password: ab1
Attempts: 1396
Time Taken: 0.00 seconds
┌──(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
└─$
KALI
BY OFFENSIVE SECURITY

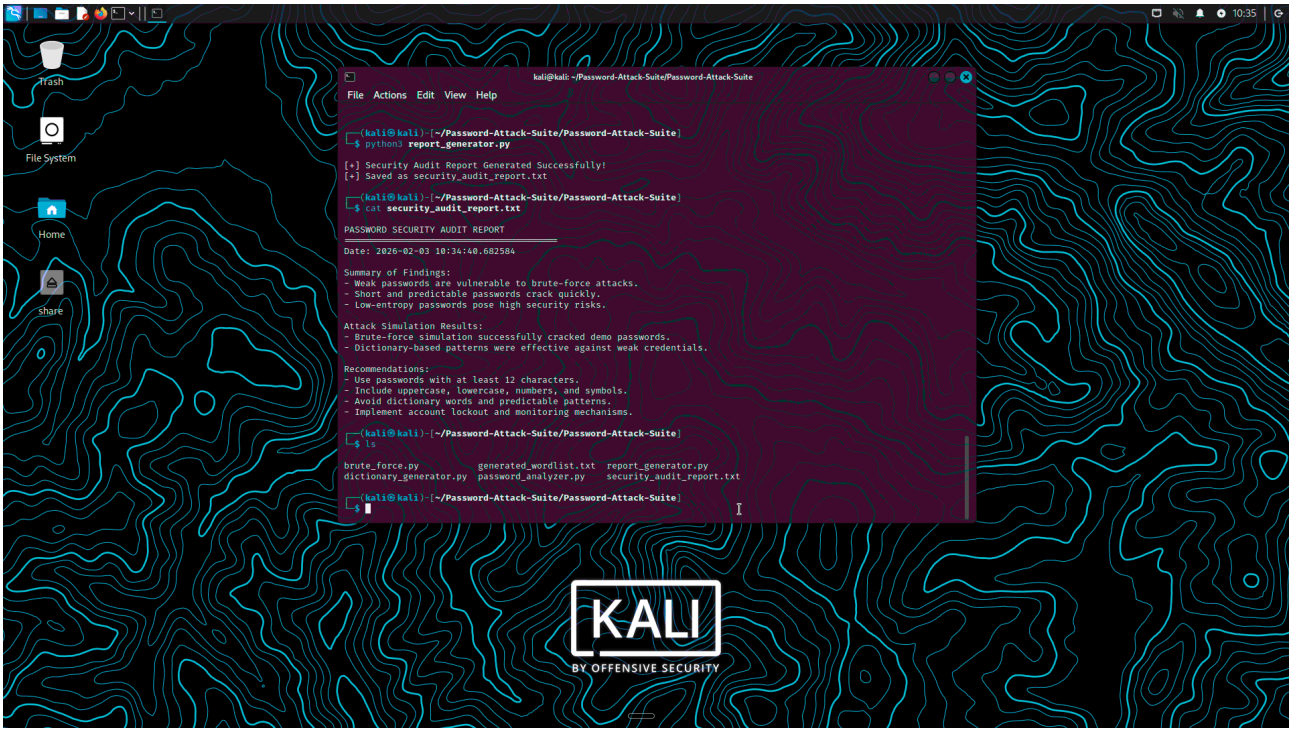

Trash
File System
Home
share
kali@kali: ~/Password-Attack-Suite/Password-Attack-Suite
File Actions Edit View Help
@dmin
test2000
User
t3$t
User2000
User1999
Test1999
┌──(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
└─$ nano brute_force.py
┌──(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
└─$ python3 brute_force.py
[*] Starting brute-force simulation...
[+] Password Found!
Password: ab1
Attempts: 1396
Time Taken: 0.00 seconds
┌──(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
└─$ nano password_analyzer.py
┌──(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
└─$ python3 password_analyzer.py
Enter password to analyze: password
[+] Password Analysis Result
Password Strength: WEAK
Estimated Entropy: 37.60 bits
Recommendation: Use longer passwords with mixed characters.
┌──(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
└─$
KALI
BY OFFENSIVE SECURITY


Trash
File System
Home
share
kali@kali: ~/Password-Attack-Suite/Password-Attack-Suite
File Actions Edit View Help
┌──(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
└─$ python3 brute_force.py
[*] Starting brute-force simulation...
[+] Password Found!
Password: ab1
Attempts: 1396
Time Taken: 0.00 seconds
┌──(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
└─$ nano password_analyzer.py
┌──(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
└─$ python3 password_analyzer.py
Enter password to analyze: password
[+] Password Analysis Result
Password Strength: WEAK
Estimated Entropy: 37.60 bits
Recommendation: Use longer passwords with mixed characters.
┌──(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
└─$ python3 password_analyzer.py
Enter password to analyze: Admin@1234
[+] Password Analysis Result
Password Strength: VERY STRONG
Estimated Entropy: 65.55 bits
┌──(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
└─$
KALI
BY OFFENSIVE SECURITY


kali@kali: ~/Password-Attack-Suite/Password-Attack-Suite
File Actions Edit View Help
GNU nano 7.2 report_generator.py *
#!/usr/bin/env python3
"""
Security Audit Report Generator
Author: AVK
Description:
Generates a final security audit report based on password analysis results.
"""
from datetime import datetime
def generate_report():
report = []
report.append("PASSWORD SECURITY AUDIT REPORT")
report.append("=" * 40)
report.append(f"Date: {datetime.now()}")
report.append("\nSummary of Findings:")
report.append("- Weak passwords are vulnerable to brute-force attacks.")
report.append("- Short and predictable passwords crack quickly.")
report.append("- Low-entropy passwords pose high security risks.")
report.append("\nAttack Simulation Results:")
report.append("- Brute-force simulation successfully cracked demo passwords.")
report.append("- Dictionary-based patterns were effective against weak credentials.")
report.append("\nRecommendations:")
report.append("- Use passwords with at least 12 characters.")
report.append("- Include uppercase, lowercase, numbers, and symbols.")
report.append("- Avoid dictionary words and predictable patterns.")
report.append("- Implement account lockout and monitoring mechanisms.")
return "\n".join(report)
if __name__ == "__main__":
final_report = generate_report()
with open("security_audit_report.txt", "w") as file:
file.write(final_report)
print("[+] Security Audit Report Generated Successfully!")
print("[+] Saved as security_audit_report.txt")
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-E Redo
KALI
BY OFFENSIVE SECURITY


kali@kali: ~/Password-Attack-Suite/Password-Attack-Suite
File Actions Edit View Help
Attempts: 1396
Time Taken: 0.00 seconds
(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
$ nano password_analyzer.py
(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
$ python3 password_analyzer.py
Enter password to analyze: password
[+] Password Analysis Result
Password Strength: WEAK
Estimated Entropy: 37.60 bits
Recommendation: Use longer passwords with mixed characters.
(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
$ python3 password_analyzer.py
Enter password to analyze: Admin@1234
[+] Password Analysis Result
Password Strength: VERY STRONG
Estimated Entropy: 65.55 bits
(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
$ nano report_generator.py
(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
$ python3 report_generator.py
[+] Security Audit Report Generated Successfully!
[+] Saved as security_audit_report.txt
(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
$
KALI
BY OFFENSIVE SECURITY

kali@kali: ~/Password-Attack-Suite/Password-Attack-Suite
File Actions Edit View Help
[+] Password Analysis Result
Password Strength: VERY STRONG
Estimated Entropy: 65.55 bits
(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
$ nano report_generator.py
(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
$ python3 report_generator.py
[+] Security Audit Report Generated Successfully!
[+] Saved as security_audit_report.txt
(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
$ cat security_audit_report.txt
PASSWORD SECURITY AUDIT REPORT
Date: 2026-02-03 10:34:40.682584
Summary of Findings:
- Weak passwords are vulnerable to brute-force attacks.
- Short and predictable passwords crack quickly.
- Low-entropy passwords pose high security risks.
Attack Simulation Results:
- Brute-force simulation successfully cracked demo passwords.
- Dictionary-based patterns were effective against weak credentials.
Recommendations:
- Use passwords with at least 12 characters.
- Include uppercase, lowercase, numbers, and symbols.
- Avoid dictionary words and predictable patterns.
- Implement account lockout and monitoring mechanisms.
(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
$
KALI
BY OFFENSIVE SECURITY


kali@kali: ~/Password-Attack-Suite/Password-Attack-Suite
File Actions Edit View Help
(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
$ python3 report_generator.py
[+] Security Audit Report Generated Successfully!
[+] Saved as security_audit_report.txt
(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
$ cat security_audit_report.txt
PASSWORD SECURITY AUDIT REPORT
Date: 2026-02-03 10:34:40.682584
Summary of Findings:
- Weak passwords are vulnerable to brute-force attacks.
- Short and predictable passwords crack quickly.
- Low-entropy passwords pose high security risks.
Attack Simulation Results:
- Brute-force simulation successfully cracked demo passwords.
- Dictionary-based patterns were effective against weak credentials.
Recommendations:
- Use passwords with at least 12 characters.
- Include uppercase, lowercase, numbers, and symbols.
- Avoid dictionary words and predictable patterns.
- Implement account lockout and monitoring mechanisms.
(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
$ ls
brute_force.py generated_wordlist.txt report_generator.py
dictionary_generator.py password_analyzer.py security_audit_report.txt
(kali㉿kali)-[~/Password-Attack-Suite/Password-Attack-Suite]
$
Trash
File System
Home
share
10:35
KALI
BY OFFENSIVE SECURITY