Module Code & Module Title

CC5009NI Cyber Security in Computing

Assessment Weightage & Type

60% Group Coursework 02

Year and Semester

2024 -25 Autumn Semester

Student Name: Sobika Pradhan London Met ID: 23047597
Student Name: Anupam Shrestha London Met ID: 23047592
Student Name: Sittal Karki London Met ID: 23047579

Assignment Due Date: May 11, Sunday

Assignment Submission Date: May 12, Monday

Word Count (Where Required):6507

23047597 Sobika Pradhan.docx

Islington College,Nepal

Document Details

Submission ID trn:oid:::3618:95273768	39 Pages
Submission Date May 11, 2025, 7:46 AM GMT+5:45	6,469 Words
Download Date May 11, 2025, 7:51 AM GMT+5:45	37,362 Characters
File Name 23047597 Sobika Pradhan.docx	
File Size 43.7 KB	

6% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

- 33 Not Cited or Quoted 5%
  Matches with neither in-text citation nor quotation marks
- 3 Missing Quotations 0%
  Matches that are still very similar to source material
- 0 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

3%	Internet sources
0%	Publications
5%	Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

Match Groups

- **33** Not Cited or Quoted 5%
  Matches with neither in-text citation nor quotation marks
- **3** Missing Quotations 0%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

3%	Internet sources
0%	Publications
5%	Submitted works (Student Papers)

Abstract

This report gives a deep analysis of brute force attacks while giving more attention to dictionary attacks which are now one of the most popular and effective attacks. cyber attackers use it to infiltrate ineffective authentication systems. The research focuses on the history, working process and the real-life effect of the dictionary attacks on the great cases, such as the data breach of LinkedIn and Adobe. In a controlled lab environment, simulation and Penetration Testing Execution Standard (PTES) framework are used to conduct a simulated attack, using practical tools such as Flask, Nmap and Wfuzz. The demonstration indicates how light weight password guessing from predefined word lists will crack login systems lacking security.

In response to such incidents, this report presents a wide-ranging assessment of active and passive defense solutions. Active techniques include the Multi-Factor Authentication (MFA), account lockout policies, CAPTCHA verification, and rate limiting; all these techniques significantly decrease automated login success rate. Other passive strategies namely, the enforcement of strict password policies and the password storage protecting through the use of hashing algorithms like bcrypt or argon2 are discussed in detail. Moreover, the significance of constant surveillance, alerting methods, and the teaching of users is emphasized to enhance the long term security posture.

The evaluation section critically analyzes the effectiveness and vulnerability of suck counter measures. While MFA and password strength policies are good defense, they can be a usability issue or cause user resistance, Similarly, CAPTCHA and rate limiting can hinder accessibility or bypassed by advanced bots. Despite suck problems, the report assumes that multi-layer design of security is still the best defense against dictionary attacks. Through the adoption of technical protection alongside awareness and regular system scanning, companies are able to sufficiently reduce their vulnerability to brute force attacks, The study brings to light the fact that proactive as well as reactive cyber-security solutions play a critical role in securing digital assts in a fast-changing threat landscape.

Table of Contents

List of figures

List of tables

1 Introduction

As in the fast-changing cyber security field, companies are often being threatened by attackers wanting to exploit the vulnerabilities of network security and information system. If any technology is gaining pace, then such technique is also advancing in cyber criminals who keep on developing new ways to break through into defences and get unauthorized entry.

Brute force attacks are among these threats because they are one of the most harmful and strongest threats to compromise exfiltrate confidential data. The concern about brute force attacks lies with their absence of reliance on software vulnerabilities, but as opposed to use of weak authentication mechanism (Owens, et al., 2008). This means that during the cyber security measures, a very strong password rules and multi factor authentication are a necessary part of any modern cyber security. Thus, it is essential to understand that the advanced methods that attackers use, and the challenging tools used have evolved. Today, brute force methods tend to use techniques such as dictionary attacks, hybrid methods and even AI assisted guessing, however, they are more flawless and harder to uncover.

This report provides an in-depth examination of a type of Brute force attack: Dictionary attack. It highlights the role of PTES in identifying and addressing dictionary attack through systematic penetration testing. Additionally, a controlled environment demonstration will assess the impact of attack and evaluate the effectiveness and application areas of current security measures.

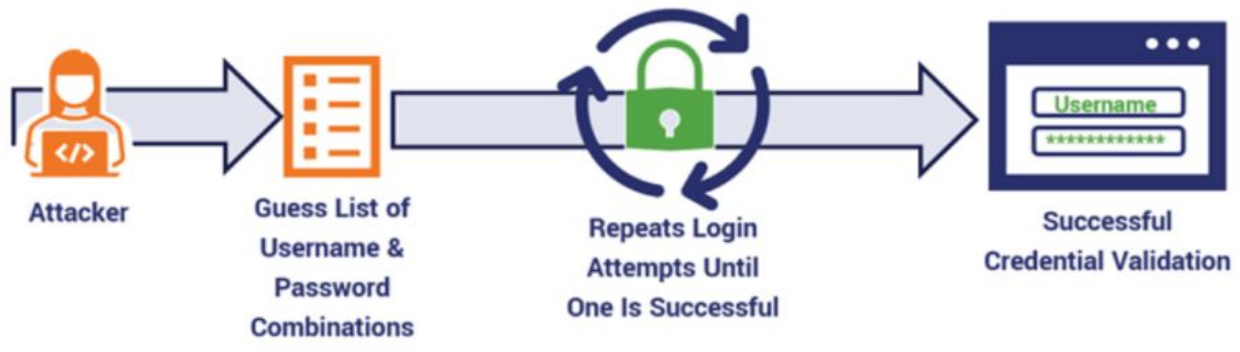

Attacker
Guess List of Username & Password Combinations
Repeats Login Attempts Until One Is Successful
Username
****
Successful Credential Validation

Figure 1: Brute Force Attack (Crane, 2021).

1.1 Report Structure

The report provides a detailed research of brute force attacks with exploring different methods, impact and growing concern in cyber security.

Section 1	Introduction	Introduction to brute force and dictionary attacks and their effect on cybersecurity.
Section 2	Background	Evolution, types and PTES standard are discussed with real world examples and required tools.
Section 3	Demonstration	Demonstration of the PTES phases is described in step by step through tools like Flask, Nmap, and Wfuzz.
Section 4	Mitigation	The attack results are then used to suggest mitigation strategies.
Section 5	Conclusion	Concluding the report with summary of key findings and recommendations for improving system resilience against dictionary attack.

Table 1: Report structure

1.2 Aim and Objective

Aim

To investigate the mechanisms, impact, and defence strategies of dictionary attacks as a components of brute force attacks, with understanding their role in mitigation techniques and cybersecurity threats.

Objective

1. To understand brute force attack mechanism more focusing on dictionary attacks, how they operate in modern systems.
2. To evaluate how brute force attacks impacts individual, business and organization.
3. To identify preventive mechanism like MFA, strong password policies and many more.
4. To explore and suggest the modern cybersecurity defences to tackle brute force attacks in personal and corporate environment.

1.3 Technical terminologies

Key to the simulation of a dictionary attack described in this section are the following terms. It is essential to understand these terms to understand the tools, methods, and findings discussed in this report.

1. **Brute force attack:** A method of sequentially trying all the potential combinations of password to gain unauthorized access.
2. **Dictionary attack:** It is a brute force attack that applies a list of probable passwords (wordlists) to guess login credentials, faster than attempting all combinations, hence faster than the previous one.
3. **Wordlists:** A file used to automate dictionary attack that contains commonly used passwords, terms, or stolen information.
4. **Credential Stuffing:** This is a type of brute force attack that uses stolen data (username and passwords) from one platform to another platform to gain unauthorized access.
5. **Hash Function:** This is a cryptographic function which converts passwords into a fixed-size string of characters which appears random.
6. **Rainbow Table Attack:** An attack using precomputed tables of hash values to reverse hashed passwords quickly.
7. **Password spraying:** To avoid detection, this low-and-slow dictionary attack variation checks a small number of popular passwords across multiple accounts.
8. **Wfuzz:** A powerful open-source tool used for conducting brute force or dictionary attacks on login pages, network services, and other password-protected systems.
9. **Nmap:** A network scanning tool used to detect live hosts, open ports, and services on a target system which is used in penetration testing.
10. **Salting:** A technique used to add a random value to passwords before hashing them, making it more difficult for attackers to use precomputed hash tables.
11. **CAPTCHA:** A human verification challenge on login forms to prevent automated tools from carrying out brute force or dictionary attacks.
12. **Rate limiting:** A protective mechanism that limits the number of logins attempts from a user or IP address within a set time frame.
13. **Account lockout policy:** A security feature that temporarily disables user accounts after set of number of logins attempt to prevent attacks.
14. **Multi-Factor Authentication (MFA):** A security method that requires users to verify their identity using multiple forms of verification.

15. **Penetration Testing Execution Standard (PTES):** A standardized framework used in ethical hacking and penetration testing that outlines key phases like intelligence gathering, exploitation, and reporting (Fakhrul, et al., 2023).

2. Background

2.1 Brute force Attack in Information Systems

Brute-force attack is one of the oldest attacks in the entire history of cybersecurity. The Brute Force Attack is a very serious risk to cyber security, as it is a testing method that is used by attackers in order to convert password, encryption keys and sensitive data and testing all possible combinations in a sequential manner until the correct one is found (Ayankoya, et al., 2019).

This method brings about constant data spills, monetary misfortunes, and services breakers (illegal access to records and frameworks). The attack is simple, and today's technologies are strong enough, but this attack is dangerous to both individual and businesses. To define brute force attacks simply, it is called password guessing but may be dictionary or credential attack and many more. (Bosnjak, et al., 2018)

2.2 The Evolution of Brute Force Attacks

The idea of brute force itself is old, but real computing brute force attempts began in the **1960s–70s** and took off in real time in the **1980s–90s** as cracking tools and computing power matured (Curtin & Matt, 2005). The trend of evolution of brute force attacks shows that cybercriminals continue to carry on this battle using brute force and cyber security continues to evolve along with that. For attackers, basic password guessing techniques have been a first step and more sophisticated attack modes followed because of automation tools and availability of big hacks datasets (Aslan, et al., 2023). As the verification process is not implemented well, brute force attacks are growing in security threat as they heavily exploit it. Nowadays the modern brute force attacks with AI based word prediction and distributed computing are no longer supported by traditional security solutions (Castagnaro, et al., 2024).

Brute force attacks remain an issue despite cyber security improvements because of poor methods of authentication and password reusing (Ayankoya, et al., 2019). Method of attacks are getting more complex so security systems must change to keep up with new threats. To get above security measures, cybercriminals use hybrid attacks, dictionary attacks, and credential stuffing. By having in place CAPTCHA challenges, account lockout methods, multi-factor authentication (MFA), and intrusion detection systems (IDS), organizations must constantly improve their protection techniques.

2.3 Working procedure of Authentication in Brute Force Attack

To carry out a Brute force attack, it is necessary to understand how an authentication works.

1. **Target Identification**: The attackers identify a system with an authentication mechanism like login pages, SSH, or encrypted files.
2. **Gather Authentication Details**: Information like Username format, password policy, login limits are collected through reconnaissance.
3. **Select an attack (Dictionary Attack)**: Attackers uses a list of common passwords instead of random guessing which involve passwords from a precompiled list of commonly used.
4. **Automate the Attack:** Tools like Hydra, Medusa, John the Ripper, Wfuzz and Hashcat try passwords from the list.
5. **Bypass Security**: Attacker uses IP rotation, CAPTCHA solvers, and slow brute force techniques to avoid detection.
6. **Gain Access**: Once a correct password is found, attackers log in and exploit the system. (Lashkari, et al., 2014)

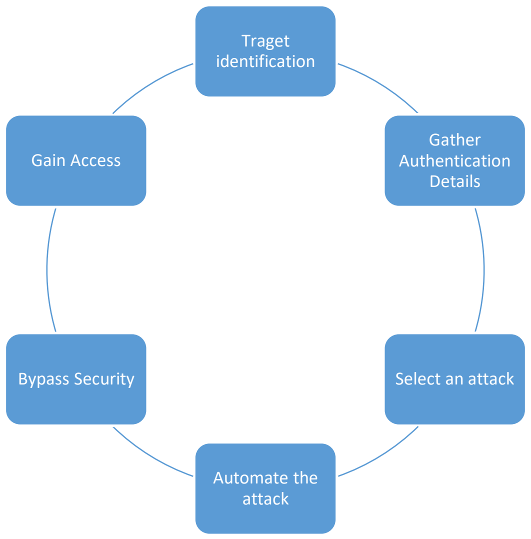


Figure 2: Working process of dictionary attack

2.4 Types of Brute Force Attack

Brute force attacks can be categorized into several types, each with its own approach and level of sophistication:

i. **Traditional Attack:** The attacker tries every character or symbol combination until the right password is found. This method is expensive and time consuming, especially for complex and long passwords.
For example, trying all combinations of numbers, letters, and symbols to crack a password. (??). (Wang, et al., 2021)

ii. **Rainbow Table Attack:** The attacker analyses passwords using precomputed tables of hash values. This method is efficient but requires significant storage for the rainbow tables.
For example, using a rainbow table to compare hashed passwords that are stored in a database. (Zhang, et al., 2017)

iii. **Dictionary Attack:** To guess the correct information, attacker uses an approved list of frequently used words, phrases or passwords. This method is faster than simple brute force because it focuses on likely passwords.
For example, using a list of common passwords like "helloworld3456","password!!" or "qwerty??". (Pinkas & Sander, 2002)

iv. **Hybrid Attack:** The attacker combines element of traditional brute force and dictionary attacks. They modify dictionary words by adding symbols, numbers to guess more complex passwords.
For example, trying "password1", "password123" or "p@ssword". (Osuagwa, 2024)

v. **Credential stuffing:** It is also called password stuffing. This is specialized form of brute force attack that uses stolen data from one platform to gain unauthorized access to other platforms.
For example, using data which was leaked from a social media platform to log into a banking website. (Ba, et al., 2021)

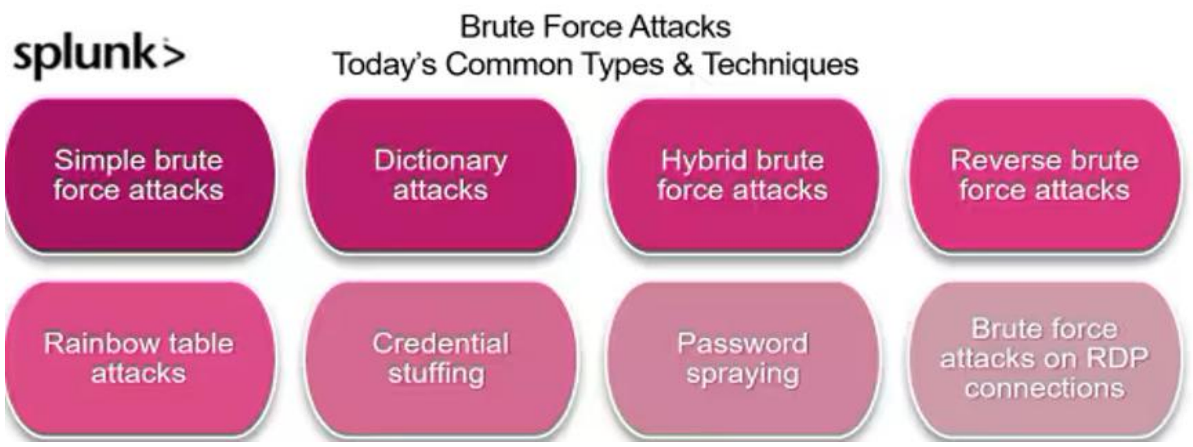

splunk>
Brute Force Attacks
Today's Common Types & Techniques
Simple brute force attacks
Dictionary attacks
Hybrid brute force attacks
Reverse brute force attacks
Rainbow table attacks
Credential stuffing
Password spraying
Brute force attacks on RDP connections

Figure 3: Several types of Brute Force Attack (Wickramasinghe, 2024).

Brute force attacks could differ in complexity from simple guessing to more complex ones like rainbow tables, dictionary assaults, and credential stuffing.

2.5 Focusing on Dictionary Attacks

Among these, Dictionary attack has become a more dangerous form of brute force attack, which is effective at breaching accounts with easily assumed credentials because they target passwords that are made up of popular words, phrases, and predictable variations (Bosnjak, et al., 2018). Dictionary attack is the most dangerous and efficient way of password cracking in which human predictability is used.

In Dictionary attacks all the passwords are tried from a precompiled list of commonly used passwords that would significantly decrease the time for guessing an account. Attackers test thousands of likely passwords in seconds with such as password file RockYou.txt and SecLists. Lots of users choose such weak and predictable passwords such as "password123", "admin" which are simple targets. Attackers further enhance these methods by using advanced tools to apply rule-based modification such as adding number, capitalizing letters or changing characters for the common password pattern.

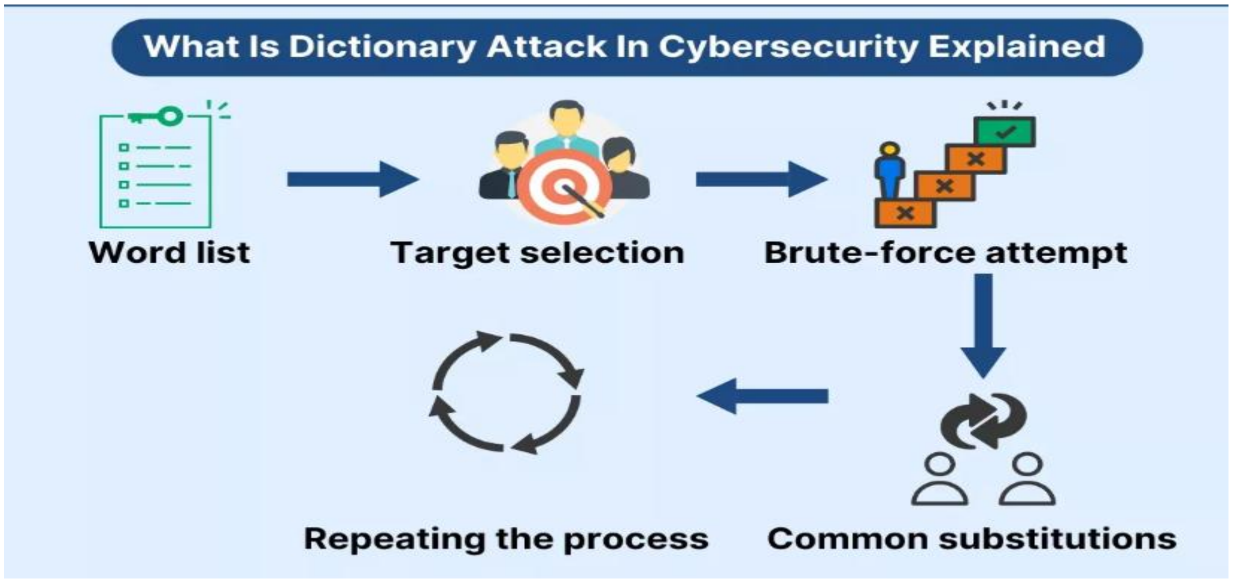


Figure 4: Dictionary Attacks (Unstop, 2024)

Then, the process of efficiency of these dictionary attacks is further enhanced by automation and AI, which uses as predicted password variations by common patterns. These attacks are a serious threat to web privacy because they can take advantage of the capacity of individuals to choose simple terms, popular words as passwords, although they are a subset of the wider brute force categories. Unlike pure brute-force attacks that attempt every possible combination, dictionary attacks focus on likely passwords, making them faster and more efficient (Alaa & AL-Shareefi, 2024).

Users, in managing dictionary attacks, are forced to adopt strong, unique passwords, as well as multi factor authentication (MFA). To prevent such attacks from causing any harm, at infrastructure there should be implemented rate limiting, intrusion detection system and account lockout policy.

2.6 Real-World Examples

LinkedIn Data Breach (2012 and 2016)

In 2012, LinkedIn saw its passwords up for grabs when 6.5 million password hashes leaked online. The stolen passwords were hashed using SHA-1 hashing not salted, so they were subject to dictionary as well as brute force attacks. A large portion of the leaked hashes was cracked by attackers quickly with precomputed wordlists and rainbow tables. It then worsened in 2016 when an expanded breach showed that 167 million LinkedIn user credentials were compromised and sold on the dark web. This attack showed the weakness of hash algorithms and the failure of salting when it comes to password databases to dictionary based cracking attempts. It means that LinkedIn caused users to reset passwords, more secure encryption and promoting multi factor authentication (MFA) to counter the future attacks. (Ramesh & Lan, 2024)

Adobe Data Breach (2013)

Adobe experienced a massive data breach as its accounts were compromised of over 153 million users in 2013. The database leaked email addresses, encrypted passwords, some with weak encryption (3DES) and none with salting. Many of the encrypted passwords could be cracked by attackers using dictionary attacks or known password patterns. Security experts were even able to use reused pattern like “photoshop” and “adobe123.” An attack occurred which highlighted the risks present in using predictable passwords and poor usability of password protection because attackers could recover actual user credentials. Users were also forced to log in again, passwords reset, and encryption improved by Adobe**.** (Marvin, 2013)

2.7 Applying the Penetration Testing Execution Standard (PTES) to Dictionary Attacks

Penetration Testing Execution Standard (PTES) is a full-fledged framework which gives a structured approach for performing penetration tests. It reviews the study of cyber-attacks, organization security and identification of vulnerabilities. (Duffy, 2015)

The PTES is useful in that it gives a structured, repeatable, clear methodology for penetration testing to keep everything as consistent as possible and as thorough as possible. It assists organizations to identify and fix vulnerabilities more clearly keeping the ethical and legal standards to be followed. The Penetration Testing Execution Standard (PTES) provides a structured approach to security testing, including dictionary attacks. In penetration testing, dictionary attacks are typically executed during the Exploitation phase of PTES, after identifying a login form during Intelligence Gathering and validating that brute-force protections like CAPTCHA or account lockout mechanisms are not in place.

Below are the key PTES phases and how they apply to dictionary attacks:

Steps	Topics	Description
1	Pre-engagement	Define scope, goals, permissions, and legal boundaries
2	Information Gathering	Find login page, password policy
3	Threat Modelling	Identify attack surface, risks
4	Vulnerability Analysis	Detect lack of CAPTCHA/MFA
5	Exploitation	Run Wfuzz with wordlist
6	Post-Exploitation	Gain access, change password
7	Reporting	Document findings and risks

Table 2: Visual representation of the PTES stages aligned with the dictionary attack process.

2.7.1 Pre-engagement Interactions (Scoping and Rules of Engagement)

Before the penetration test begins, agreements, objectives, scope, and legal permissions are clearly discussed.

- **Scope Definition:** Define which systems and authentication mechanisms are going to be tested through forms like SSH login portals, web-based login forms or database admin panels.
- **Rules of Engagement:** Specify expectations of those involved in the test including/outside of business hours, how many login attempts are allowable without disruption.
- **Authorization:** Official written permission is obtained (Authorization) of the organization to test. It also includes agreement on allowable tools, allowable attack time, and allowable target. The activities of all the testing were performed under controlled lab environment and were securely authorized. Prior to testing, an agreement was made on the scope, rules of engagement, and the tools to be used. The objective was to simulate a real-world attack scenario such that the resulting situation was predictable and under proper supervision. This phase ensures mutual understanding, limits unintended consequences and maintains legal and ethical boundaries throughout the test. It also provided a foundation for responsible reporting and accountability at later stages.

2.7.2 Information Gathering

Collect information about the target using open-source tools, public records, social media, network scanning without attacking it.

Passive Reconnaissance:

- **Endpoint Identification:** Find login portals or services to which anyone connect (e.g.: SSH, FTP or even HTTP login pages).
- **Password Policy Discovery:** To begin looking at password requirements would ideally be accomplished using OSINT (Open-Source Intelligence) or previous data breaches to know exactly what to look for.
- **Data Sources:** Build or refine the dictionary list with reference to known leaked credential databases (e.g. RockYou.txt, HaveIBeenPAwned).

Active Reconnaissance:

- **User Enumeration:** Error messages or behavioural differences in the login responses may be tried to identify valid usernames.

Tool Usage: Nmap (Network Mapper): Nmap is an open-source powerful tool to do network scanning and vulnerability detection (Shah, et al., 2019). For this phase, it was used to scan an open port and running services on the target machine before the dictionary attack.

2.7.3 Threat modelling

Analyse the gathered information to understand how an attacker could harm the system, identify possible vulnerabilities and determine which components are most at risk.

- **Attack Surface Analysis:** Find out which system is threatened if the passwords are weak or reused.
- **Entry Points:** Identify endpoints that do not have enough authentication security and may enable automated password guessing.

- **Security Control Assessment:** Check to see if measures such as CAPTCHA, MFA, rate limiting, etc. are looking to perform or stop dictionary attempts.
- **Risk Impact:** Assess how a successful attack would impact (data leakage, lateral movement) of critical systems.

2.7.4 Vulnerability Analysis:

Actively scan the system for weaknesses using tools and manual techniques to find flaws that attackers might exploit. Both manual inspection and automated tools were used to identify flaws in the authentication infrastructure and security configurations.

- **Multi-Factor Authentication (MFA) Status:** Decide whether multi factor authentication (MFA) is activated. Successful credential-based attacks, despite MFA, require much less effort than an attack not requiring MFA.
- **Password Policy Weaknesses:** Check if passwords meet password policies. Enforcement of strong password policies should be evaluated by rate-limiting, minimum complexity, use of CAPTCHA, account lockout thresholds, and other features of the systems.
- **Password Storage Security:** Investigate if the system employs lame password hashing algorithms (such as unsalted MD5, SHA-1).

Tools Usage: Nmap (Network Mapper): Nmap is an open-source powerful tool to do network scanning and vulnerability detection (Shah, et al., 2019). For this phase, Nmap to **run vulnerability scripts** or version scans to look for known exploits.

2.7.5 Exploitation (Executing the Dictionary Attack)

In this stage, the goal is to attempt to exploit the vulnerabilities found (ethically), trying to gain unauthorized access, escalate privileges, or extract sensitive data.

- **Credential Testing:** Run test dictionary-based login attempts against services like SSH, FTP or web portal.

- **Control Detection:** During the attack, observations were made to detect **security controls** such as **rate limiting**, **account lockouts**, or other defenses that prevents unauthorized login attempts.

Tool Usage: Wfuzz: It's a web application brute force and fuzzing (black box) tool called Wfuzz (van Rooij, et al., 2021). Login attempts were carried out using Wfuzz, which is commonly used for brute-force attacks.

Python3: The (.py) script is run on the programming language Python 3. This script launches the Flask web app and returns the login view. In this phase, python3 is used to check attack attempts.

2.7.6 Post-Exploitation (Assessing the Impact)

After successful exploitation, assess the value of the compromised system, maintain access, extract information, and understand the business impact.

- **Privilege Escalation:** Find out what level of access was passed through a dictionary attack successful attack.
- **Data discovery:** If you have compromised administrative credentials then try to escalate your privilege.
- **Persistence:** Finally, extract and analyse further sensitive data if appropriate.

2.7.7 Reporting

This level includes capturing the entire penetration testing process, noting findings, evaluating risks and making recommendations that will guide security.

- **Report Preparation:**
  After the test, a comprehensive report is generated that includes document findings that is successful and unsuccessful attack attempts.
- **Recommendations:**
  The first step is to recommend security improvement such as stronger password policies, MFA, Account Lockout features and rate limiting.
- **Risk Assessment:**
  Prepare a risk assessment report with how the attack can be mitigated in the future.

- **Framework Alignment:**
  PTES provides penetration testers with a systematic guide to evaluate and improve the authentication security against dictionary attacks in an ethical and legal manner.

3. Demonstration

This section shows that how to do the brute force attack on custom web login page created using flask web framework. The purpose of this demonstration is to apply the theoretical concepts of brute force and dictionary attacks in actual controlled setup.

These steps follow the Penetration Testing Execution Standard (PTES), a structured methodology used for conducting ethical penetration tests. A local fake mobile bank login system was created to simulate a real-world authentication environment. The tools which were used are Nmap and Wfuzz, Used for scanning and fuzzing input fields respectively. Pre created wordlist of username.txt and password.txt are used to test the username and password and verify successful login attempts.

The demonstration is broken down into each step and are arranged in PTES phase, from initial reconnaissance all the way to post exploit analysis, allowing to review how dictionary attack is done in step by step, ethical testing scenario.

Lab setup

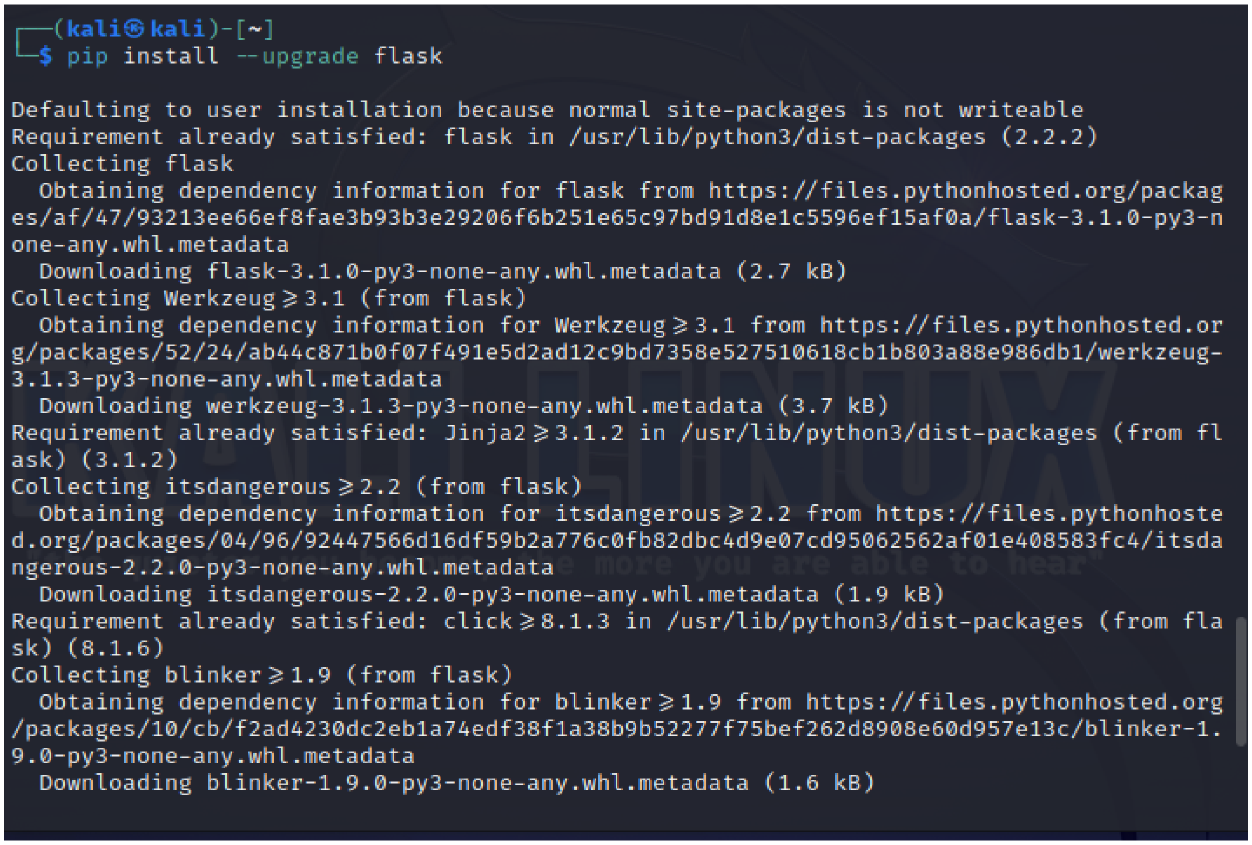
For demonstration, controlled lab setup was created using a custom flask web application with a login form vulnerable to dictionary attacks. Tools like flask and Wfuzz were installed locally, and a test wordlist was prepared.

Step 1: Installing or upgrading flask.

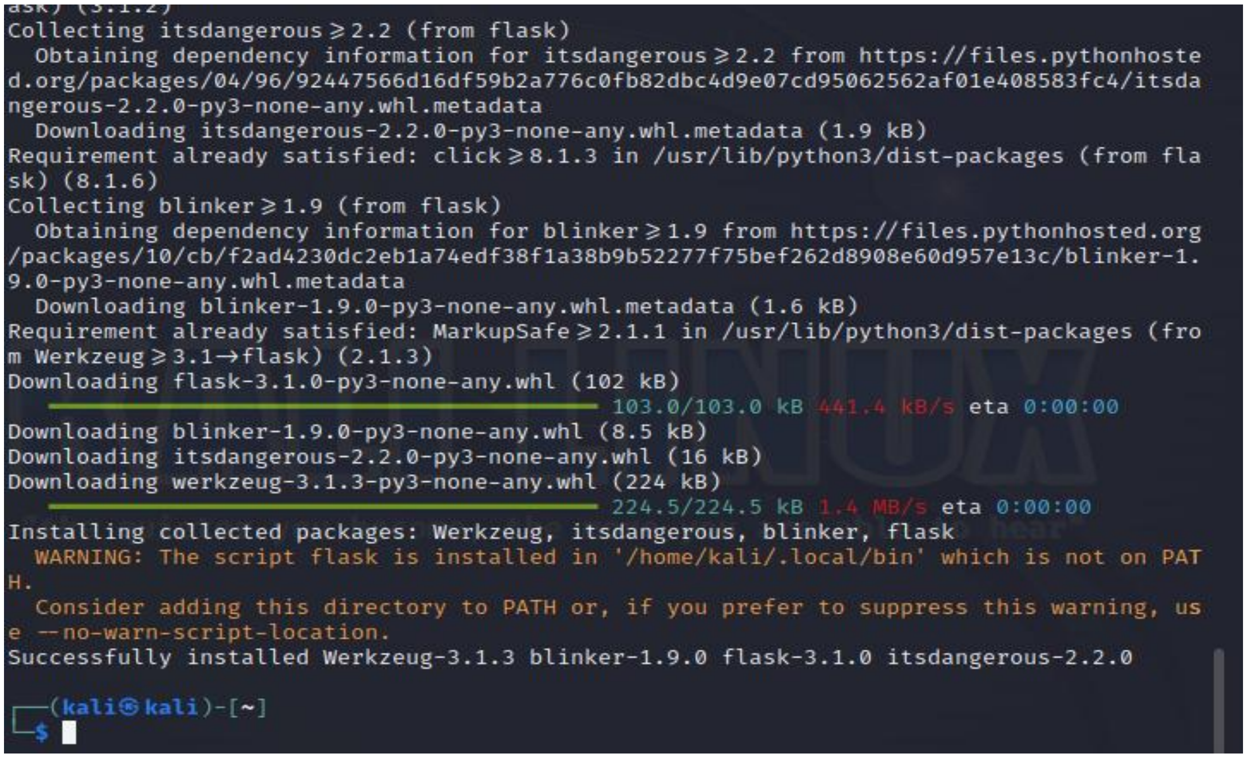
Flask: Flask is a lightweight web framework written in Python, which provides easy setup of web server and servers with the same environment for the login as a real-world environment for authentication testing (Aslam, et al., 2015).

Command used is **Pip install - -upgrade flask.**

Here, Pip: Python's package installer and - -upgrade: It tells pip to upgrade the package to latest newer version. It checks currently installed version of Flask and compares it to the latest version available on PyPI.



```
┌──(kali㉿kali)-[~]
└─$ pip install --upgrade flask

Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: flask in /usr/lib/python3/dist-packages (2.2.2)
Collecting flask
  Obtaining dependency information for flask from https://files.pythonhosted.org/packag
es/af/47/93213ee66ef8fae3b93b3e29206f6b251e65c97bd91d8e1c5596ef15af0a/flask-3.1.0-py3-n
one-any.whl.metadata
  Downloading flask-3.1.0-py3-none-any.whl.metadata (2.7 kB)
Collecting Werkzeug≥3.1 (from flask)
  Obtaining dependency information for Werkzeug≥3.1 from https://files.pythonhosted.or
g/packages/52/24/ab44c871b0f07f491e5d2ad12c9bd7358e527510618cb1b803a88e986db1/werkzeug-
3.1.3-py3-none-any.whl.metadata
  Downloading werkzeug-3.1.3-py3-none-any.whl.metadata (3.7 kB)
Requirement already satisfied: Jinja2≥3.1.2 in /usr/lib/python3/dist-packages (from fl
ask) (3.1.2)
Collecting itsdangerous≥2.2 (from flask)
  Obtaining dependency information for itsdangerous≥2.2 from https://files.pythonhoste
d.org/packages/04/96/92447566d16df59b2a776c0fb82dbc4d9e07cd95062562af01e408583fc4/itsda
ngerous-2.2.0-py3-none-any.whl.metadata
  Downloading itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9 kB)
Requirement already satisfied: click≥8.1.3 in /usr/lib/python3/dist-packages (from fla
sk) (8.1.6)
Collecting blinker≥1.9 (from flask)
  Obtaining dependency information for blinker≥1.9 from https://files.pythonhosted.org
/packages/10/cb/f2ad4230dc2eb1a74edf38f1a38b9b52277f75bef262d8908e60d957e13c/blinker-1.
9.0-py3-none-any.whl.metadata
  Downloading blinker-1.9.0-py3-none-any.whl.metadata (1.6 kB)
```



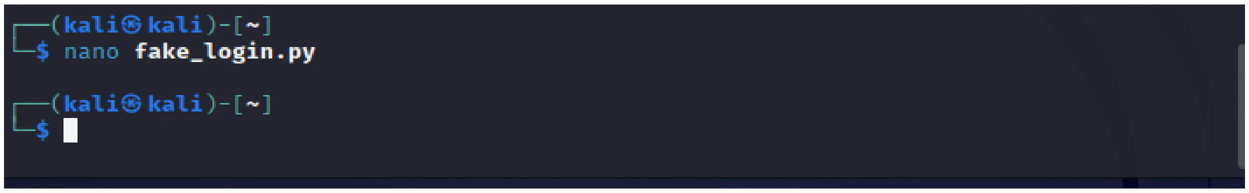
```
ask) (3.1.2)
Collecting itsdangerous≥2.2 (from flask)
  Obtaining dependency information for itsdangerous≥2.2 from https://files.pythonhoste
d.org/packages/04/96/92447566d16df59b2a776c0fb82dbc4d9e07cd95062562af01e408583fc4/itsda
ngerous-2.2.0-py3-none-any.whl.metadata
  Downloading itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9 kB)
Requirement already satisfied: click≥8.1.3 in /usr/lib/python3/dist-packages (from fla
sk) (8.1.6)
Collecting blinker≥1.9 (from flask)
  Obtaining dependency information for blinker≥1.9 from https://files.pythonhosted.org
/packages/10/cb/f2ad4230dc2eb1a74edf38f1a38b9b52277f75bef262d8908e60d957e13c/blinker-1.
9.0-py3-none-any.whl.metadata
  Downloading blinker-1.9.0-py3-none-any.whl.metadata (1.6 kB)
Requirement already satisfied: MarkupSafe≥2.1.1 in /usr/lib/python3/dist-packages (fro
m Werkzeug≥3.1→flask) (2.1.3)
Downloading flask-3.1.0-py3-none-any.whl (102 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 103.0/103.0 kB 441.4 kB/s eta 0:00:00
Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Downloading werkzeug-3.1.3-py3-none-any.whl (224 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 224.5/224.5 kB 1.4 MB/s eta 0:00:00
Installing collected packages: Werkzeug, itsdangerous, blinker, flask
  WARNING: The script flask is installed in '/home/kali/.local/bin' which is not on PAT
H.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, us
e --no-warn-script-location.
Successfully installed Werkzeug-3.1.3 blinker-1.9.0 flask-3.1.0 itsdangerous-2.2.0

┌──(kali㉿kali)-[~]
└─$ 
```

Figure 5: Installing and upgrading Flask on Kali Linux

Step 2: Making fake file and inserting data inside it.

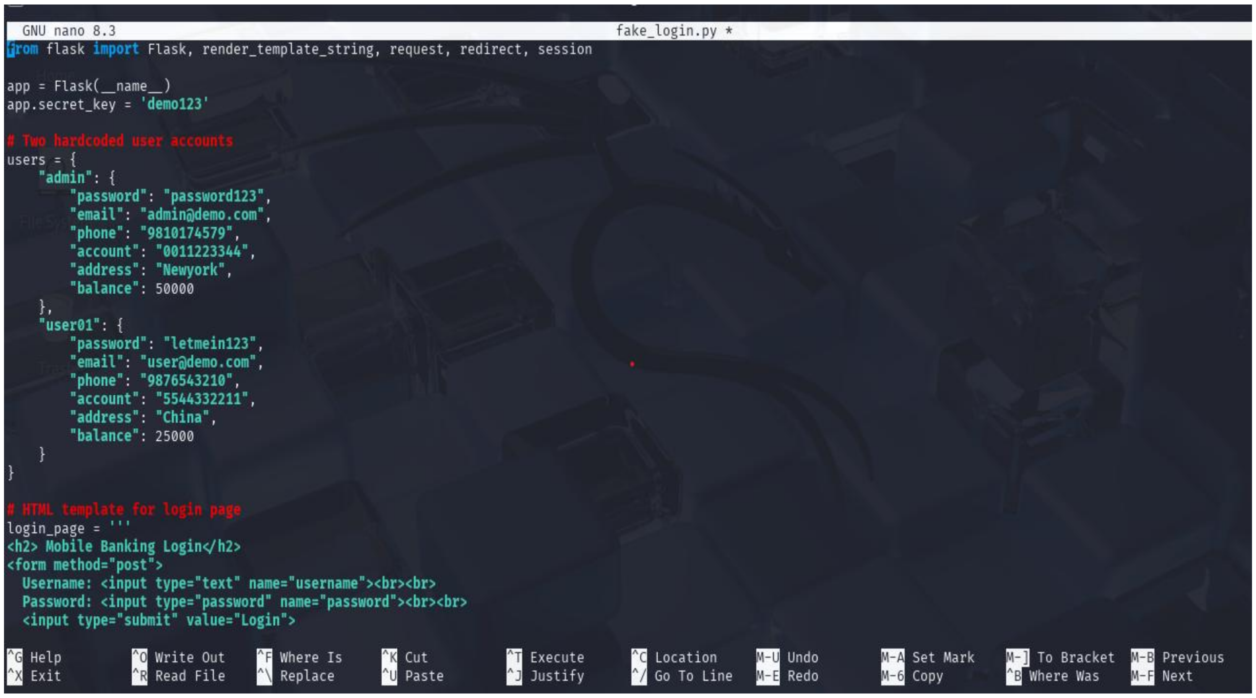
Command: **nano fake_login.py**

Nano: It is mainly a text editor but also works to make some files.

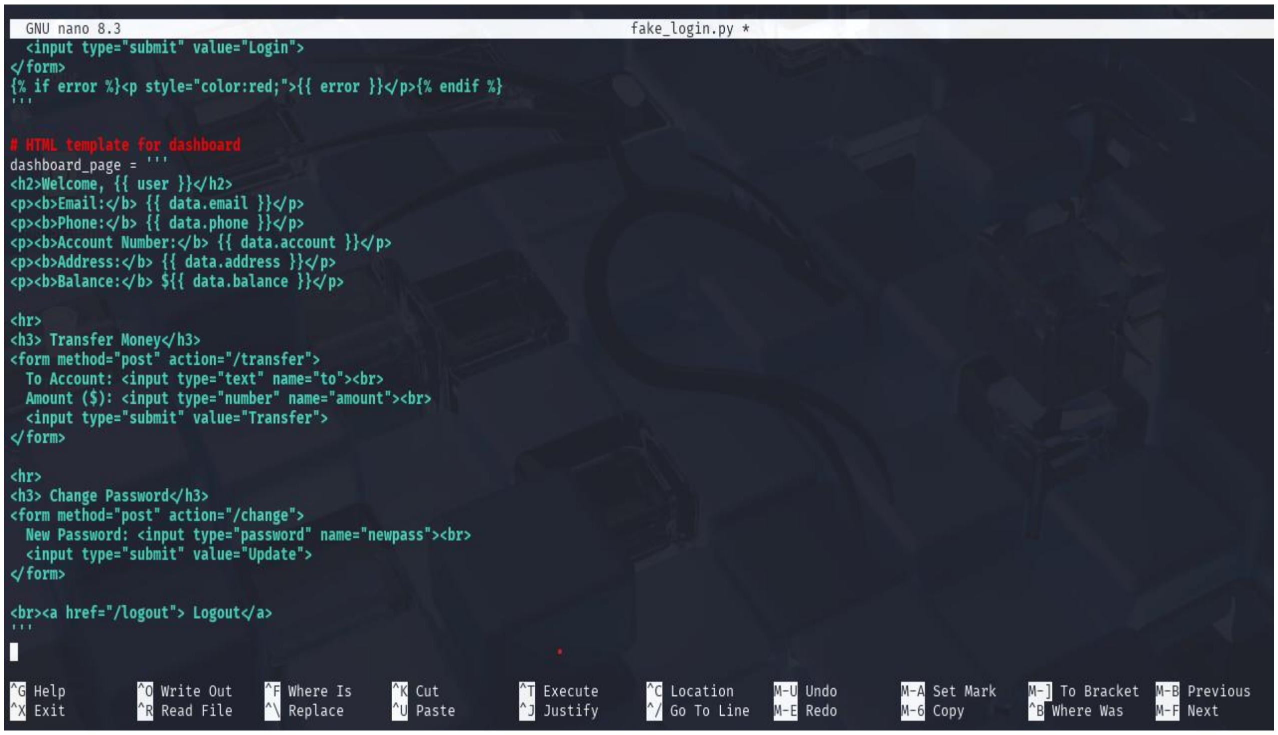


```
┌──(kali㉿kali)-[~]
└─$ nano fake_login.py

┌──(kali㉿kali)-[~]
└─$ 
```

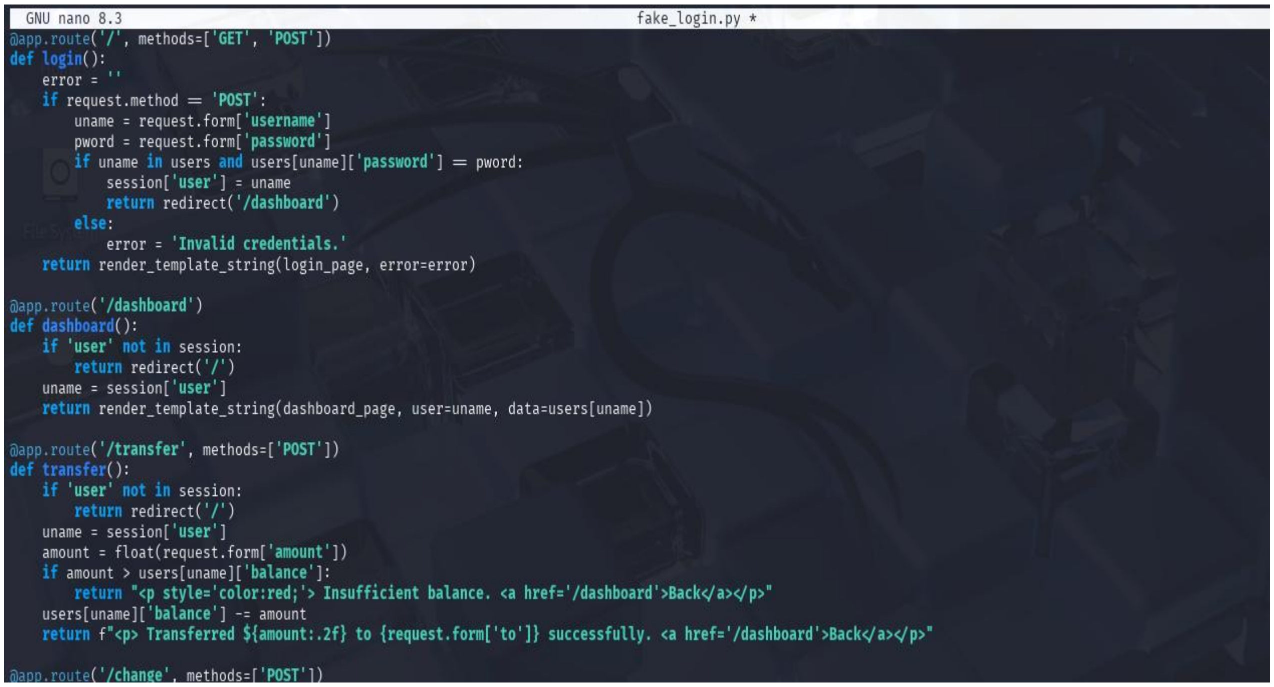
In the first line, command is: **From flask import Flask, request, render_template_string, redirect, session** this means, From flask package we are importing flask, request, render_template_string, redirect, session. Flask creates the web app. request gets data sent from the user (form inputs). render_template_string displays HTML written inside your Python code. Redirect sends the user to a different page/URL. Session stores user data (like login info) across pages. The code is written in both html and python. For display, simple html is used and for backend python is used.



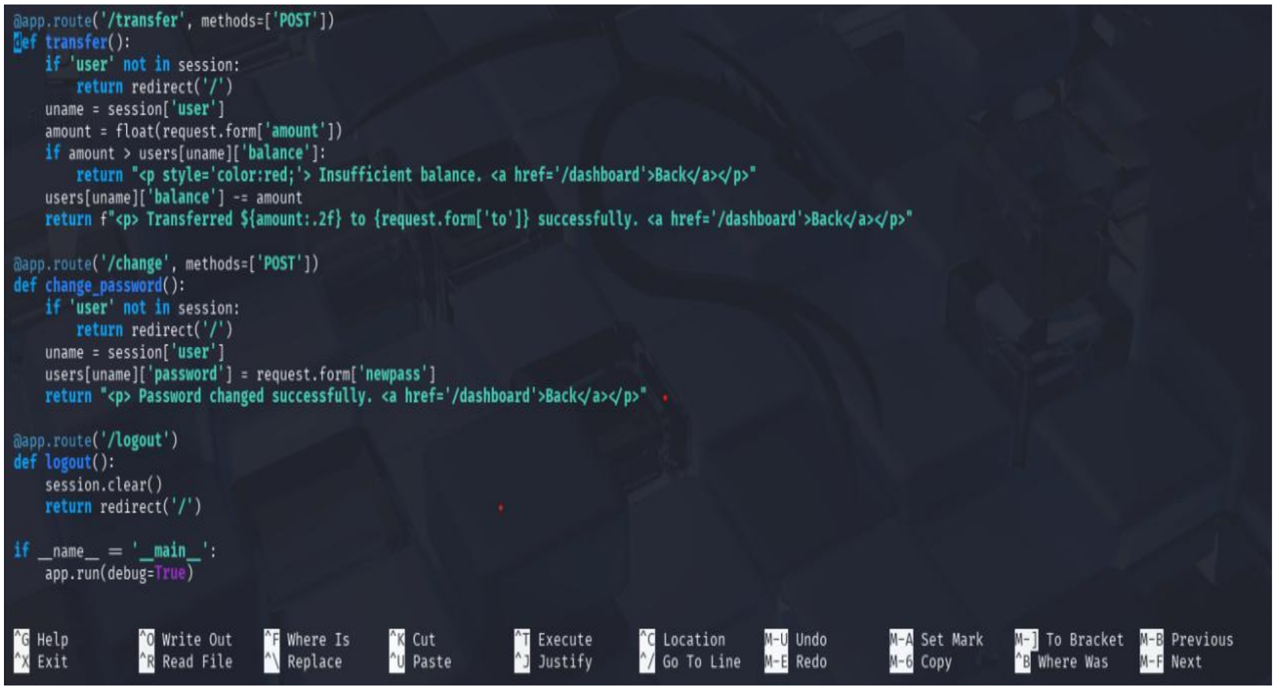
```
  GNU nano 8.3                                         fake_login.py *
from flask import Flask, render_template_string, request, redirect, session

app = Flask(__name__)
app.secret_key = 'demo123'

# Two hardcoded user accounts
users = {
    "admin": {
        "password": "password123",
        "email": "admin@demo.com",
        "phone": "9810174579",
        "account": "0011223344",
        "address": "Newyork",
        "balance": 50000
    },
    "user01": {
        "password": "letmein123",
        "email": "user@demo.com",
        "phone": "9876543210",
        "account": "5544332211",
        "address": "China",
        "balance": 25000
    }
}

# HTML template for login page
login_page = '''
<h2> Mobile Banking Login</h2>
<form method="post">
  Username: <input type="text" name="username"><br><br>
  Password: <input type="password" name="password"><br><br>
  <input type="submit" value="Login">
^G Help       ^O Write Out  ^F Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-A Set Mark  M-] To Bracket  M-B Previous
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo      M-6 Copy      ^B Where Was    M-F Next
```



```
  <input type="submit" value="Login">
</form>
{% if error %}<p style="color:red;">{{ error }}</p>{% endif %}
'''

# HTML template for dashboard
dashboard_page = '''
<h2>Welcome, {{ user }}</h2>
<p><b>Email:</b> {{ data.email }}</p>
<p><b>Phone:</b> {{ data.phone }}</p>
<p><b>Account Number:</b> {{ data.account }}</p>
<p><b>Address:</b> {{ data.address }}</p>
<p><b>Balance:</b> ${{ data.balance }}</p>

<hr>
<h3> Transfer Money</h3>
<form method="post" action="/transfer">
  To Account: <input type="text" name="to"><br>
  Amount ($): <input type="number" name="amount"><br>
  <input type="submit" value="Transfer">
</form>

<hr>
<h3> Change Password</h3>
<form method="post" action="/change">
  New Password: <input type="password" name="newpass"><br>
  <input type="submit" value="Update">
</form>

<br><a href="/logout"> Logout</a>
'''
```



```
@app.route('/', methods=['GET', 'POST'])
def login():
    error = ''
    if request.method == 'POST':
        uname = request.form['username']
        pword = request.form['password']
        if uname in users and users[uname]['password'] == pword:
            session['user'] = uname
            return redirect('/dashboard')
        else:
            error = 'Invalid credentials.'
    return render_template_string(login_page, error=error)

@app.route('/dashboard')
def dashboard():
    if 'user' not in session:
        return redirect('/')
    uname = session['user']
    return render_template_string(dashboard_page, user=uname, data=users[uname])

@app.route('/transfer', methods=['POST'])
def transfer():
    if 'user' not in session:
        return redirect('/')
    uname = session['user']
    amount = float(request.form['amount'])
    if amount > users[uname]['balance']:
        return "<p style='color:red;'> Insufficient balance. <a href='/dashboard'>Back</a></p>"
    users[uname]['balance'] -= amount
    return f"<p> Transferred ${amount:.2f} to {request.form['to']} successfully. <a href='/dashboard'>Back</a></p>"

@app.route('/change', methods=['POST'])
```

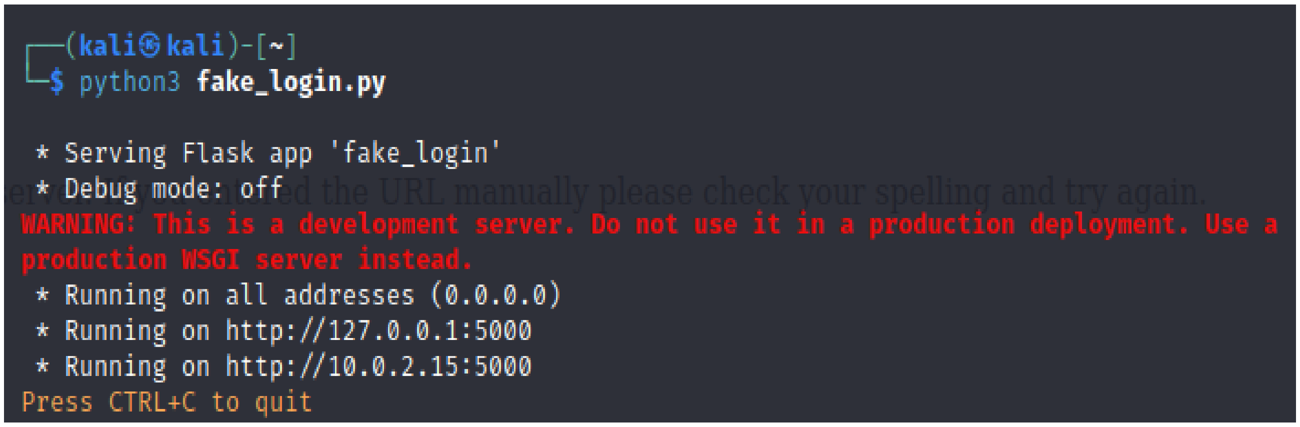


```
@app.route('/transfer', methods=['POST'])
def transfer():
    if 'user' not in session:
        return redirect('/')
    uname = session['user']
    amount = float(request.form['amount'])
    if amount > users[uname]['balance']:
        return "<p style='color:red;'> Insufficient balance. <a href='/dashboard'>Back</a></p>"
    users[uname]['balance'] -= amount
    return f"<p> Transferred ${amount:.2f} to {request.form['to']} successfully. <a href='/dashboard'>Back</a></p>"

@app.route('/change', methods=['POST'])
def change_password():
    if 'user' not in session:
        return redirect('/')
    uname = session['user']
    users[uname]['password'] = request.form['newpass']
    return "<p> Password changed successfully. <a href='/dashboard'>Back</a></p>"

@app.route('/logout')
def logout():
    session.clear()
    return redirect('/')

if __name__ == '__main__':
    app.run(debug=True)
```

Figure 6: Creating the python script to stimulate vulnerable login page

Step 3: Running the code and testing it.

Python3 is used for running Python file using **Python3**. Data will be more after we complete our attack. Command is **python3 fake_login.py**



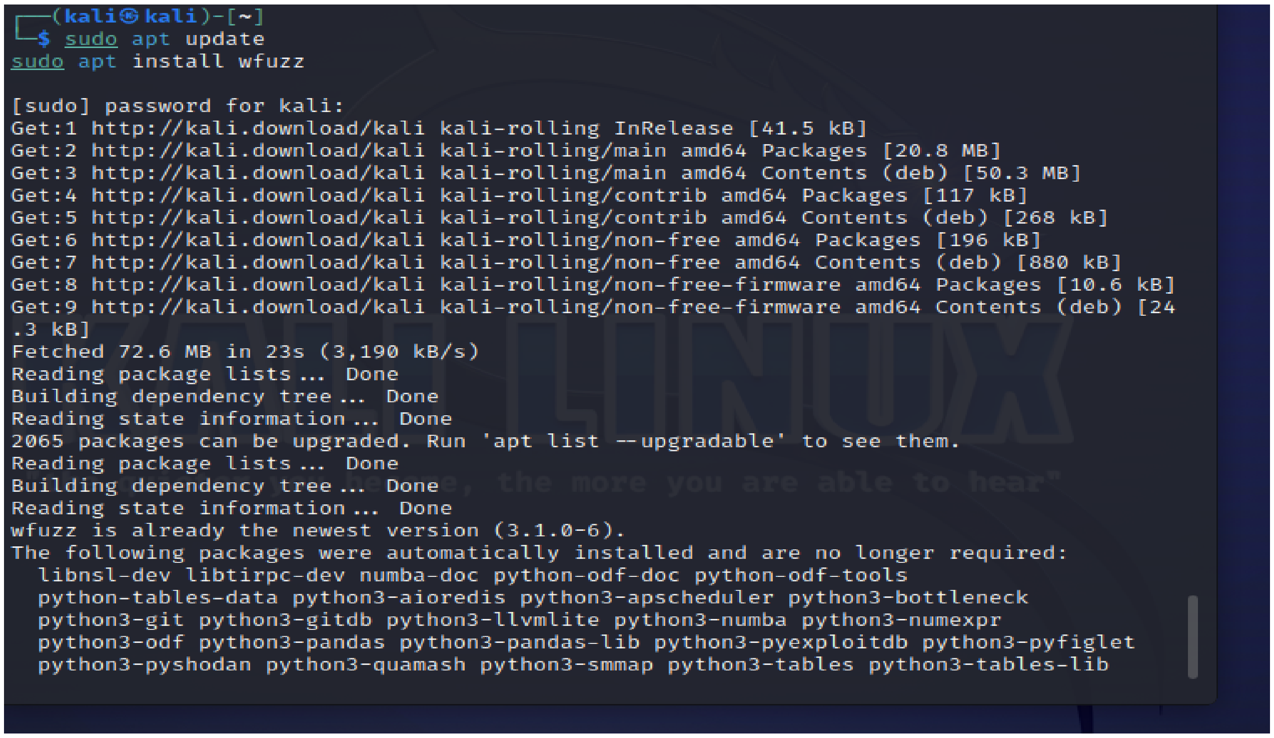
```
┌──(kali㉿kali)-[~]
└─$ python3 fake_login.py

 * Serving Flask app 'fake_login'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a
production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://10.0.2.15:5000
Press CTRL+C to quit
```

Figure 7: Using python3 to test

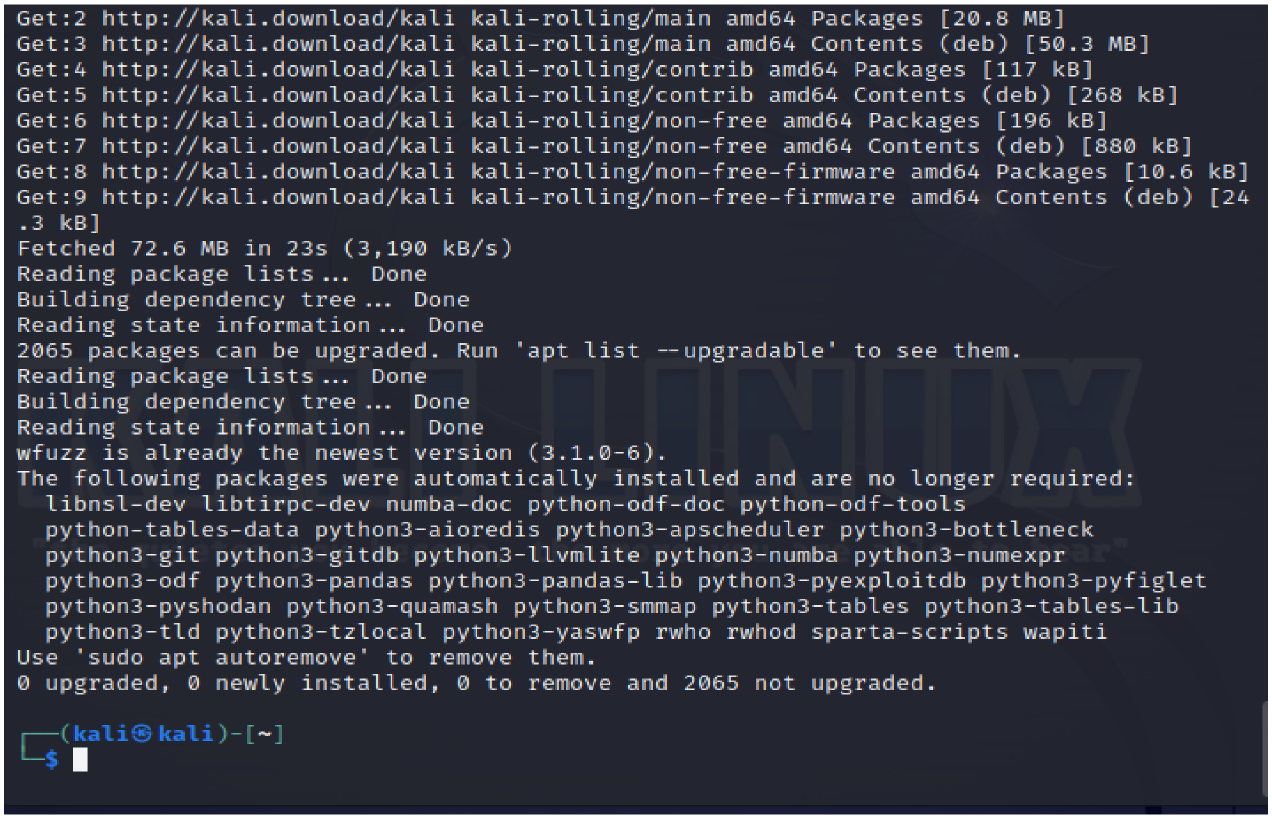
Step 4: Installing Wfuzz.

Wfuzz is a powerful web application fuzzing tool used in penetration testing to discover hidden or vulnerable elements of a web application.

Command is **sudo apt install wfuzz.**



```
┌──(kali㉿kali)-[~]
└─$ sudo apt update
sudo apt install wfuzz

[sudo] password for kali:
Get:1 http://kali.download/kali kali-rolling InRelease [41.5 kB]
Get:2 http://kali.download/kali kali-rolling/main amd64 Packages [20.8 MB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [50.3 MB]
Get:4 http://kali.download/kali kali-rolling/contrib amd64 Packages [117 kB]
Get:5 http://kali.download/kali kali-rolling/contrib amd64 Contents (deb) [268 kB]
Get:6 http://kali.download/kali kali-rolling/non-free amd64 Packages [196 kB]
Get:7 http://kali.download/kali kali-rolling/non-free amd64 Contents (deb) [880 kB]
Get:8 http://kali.download/kali kali-rolling/non-free-firmware amd64 Packages [10.6 kB]
Get:9 http://kali.download/kali kali-rolling/non-free-firmware amd64 Contents (deb) [24
.3 kB]
Fetched 72.6 MB in 23s (3,190 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
2065 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
wfuzz is already the newest version (3.1.0-6).
The following packages were automatically installed and are no longer required:
  libnsl-dev libtirpc-dev numba-doc python-odf-doc python-odf-tools
  python-tables-data python3-aioredis python3-apscheduler python3-bottleneck
  python3-git python3-gitdb python3-llvmlite python3-numba python3-numexpr
  python3-odf python3-pandas python3-pandas-lib python3-pyexploitdb python3-pyfiglet
  python3-pyshodan python3-quamash python3-smmap python3-tables python3-tables-lib
```



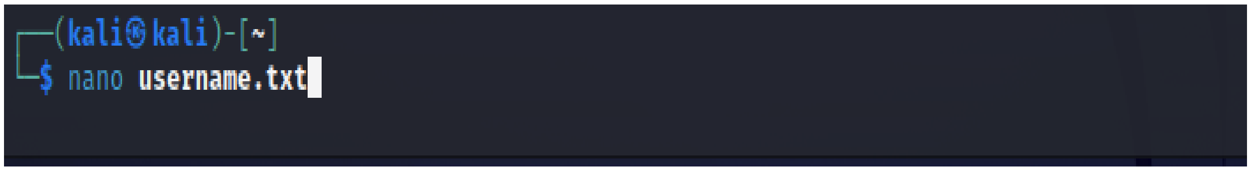
```
Get:2 http://kali.download/kali kali-rolling/main amd64 Packages [20.8 MB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [50.3 MB]
Get:4 http://kali.download/kali kali-rolling/contrib amd64 Packages [117 kB]
Get:5 http://kali.download/kali kali-rolling/contrib amd64 Contents (deb) [268 kB]
Get:6 http://kali.download/kali kali-rolling/non-free amd64 Packages [196 kB]
Get:7 http://kali.download/kali kali-rolling/non-free amd64 Contents (deb) [880 kB]
Get:8 http://kali.download/kali kali-rolling/non-free-firmware amd64 Packages [10.6 kB]
Get:9 http://kali.download/kali kali-rolling/non-free-firmware amd64 Contents (deb) [24
.3 kB]
Fetched 72.6 MB in 23s (3,190 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
2065 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
wfuzz is already the newest version (3.1.0-6).
The following packages were automatically installed and are no longer required:
  libnsl-dev libtirpc-dev numba-doc python-odf-doc python-odf-tools
  python-tables-data python3-aioredis python3-apscheduler python3-bottleneck
  python3-git python3-gitdb python3-llvmlite python3-numba python3-numexpr
  python3-odf python3-pandas python3-pandas-lib python3-pyexploitdb python3-pyfiglet
  python3-pyshodan python3-quamash python3-smmap python3-tables python3-tables-lib
  python3-tld python3-tzlocal python3-yaswfp rwho rwhod sparta-scripts wapiti
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 2065 not upgraded.

┌──(kali㉿kali)-[~]
└─$ 
```

Figure 8: Installing wfuzz.

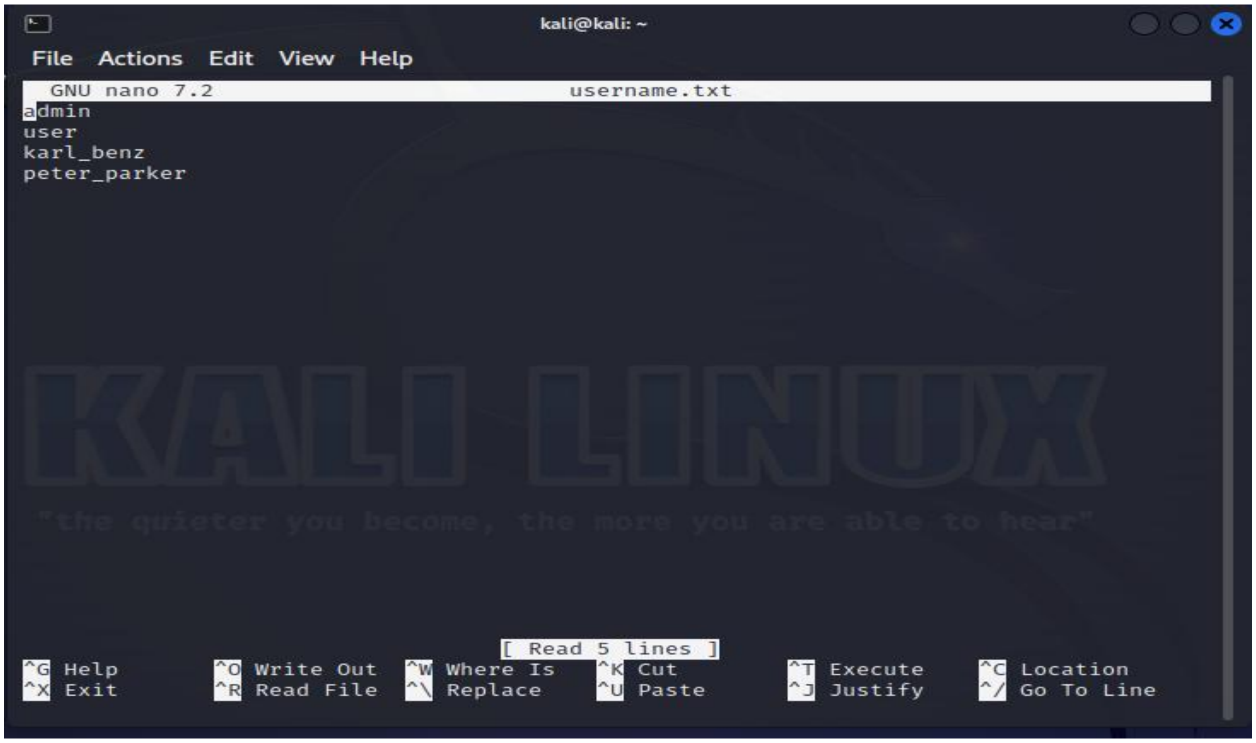
Step 5: Making different files for passwords and username and inserting probable password and usernames.

Made a file called username.txt. Probable username is inserted in that file.

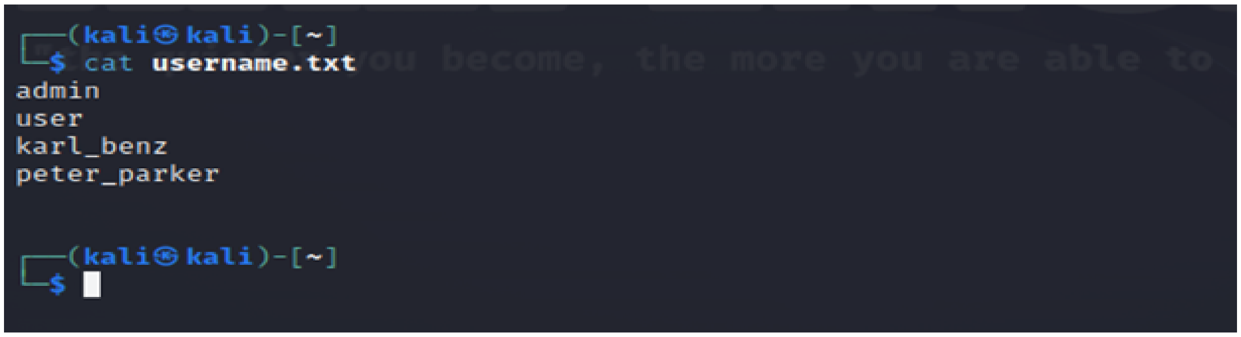
Username



```
┌──(kali㉿kali)-[~]
└─$ nano username.txt
```



```
  GNU nano 7.2                        username.txt
admin
user
karl_benz
peter_parker



                              [ Read 5 lines ]
^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line
```

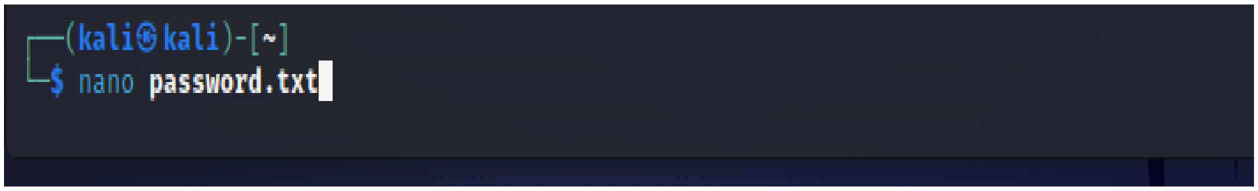


```
┌──(kali㉿kali)-[~]
└─$ cat username.txt
admin
user
karl_benz
peter_parker

┌──(kali㉿kali)-[~]
└─$ 
```

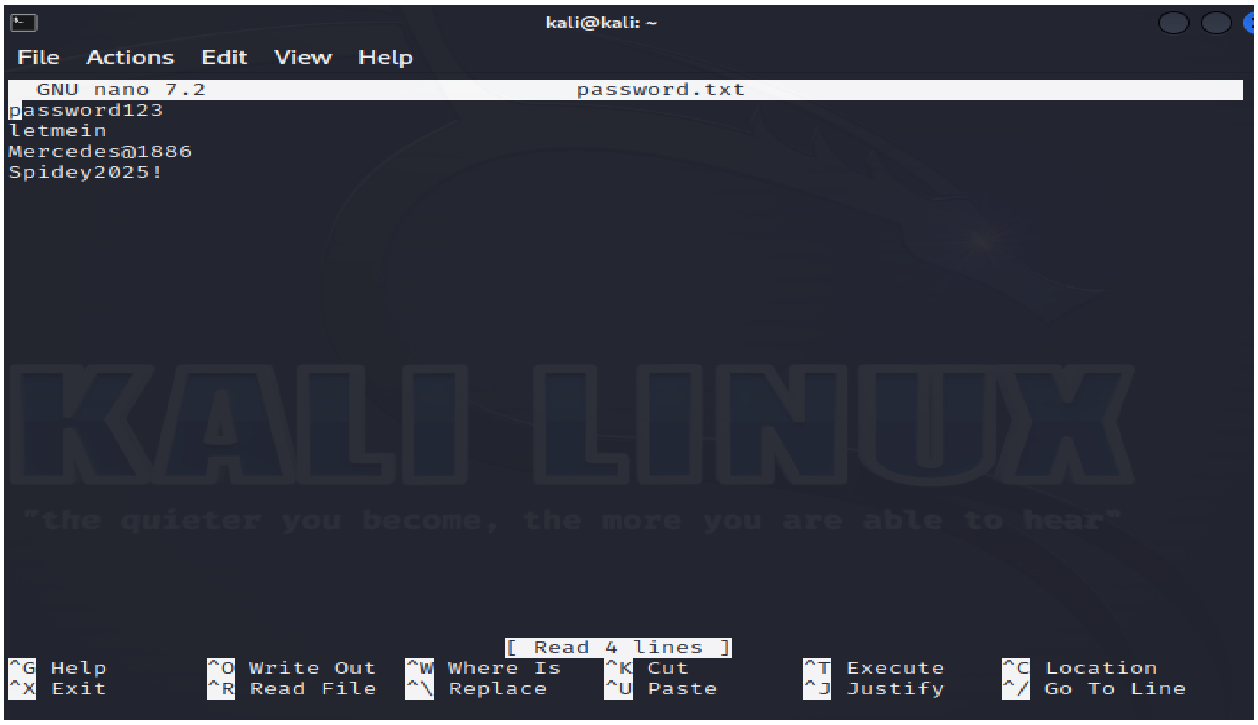
Figure 9: Creating list of probable username and checking.

Made a file called password.txt. Probable password is inserted in this file.

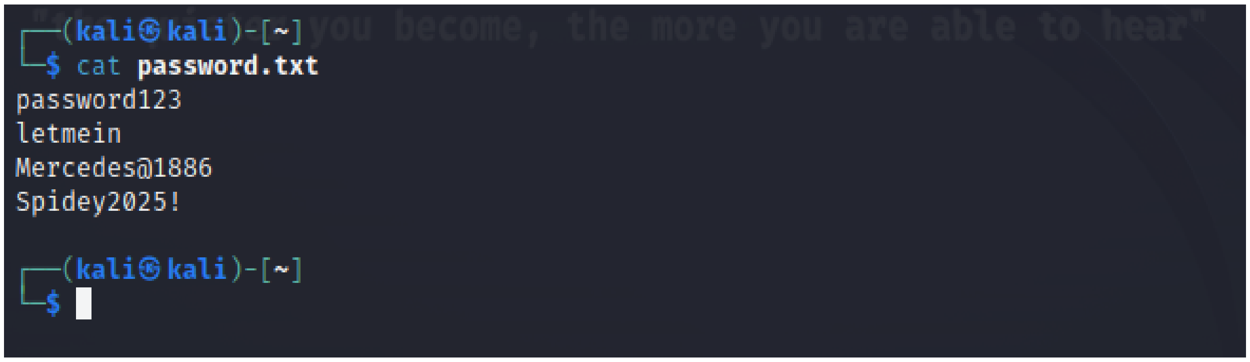
Password



```
┌──(kali㉿kali)-[~]
└─$ nano password.txt
```



```
kali@kali: ~
File  Actions  Edit  View  Help
  GNU nano 7.2                              password.txt
password123
letmein
Mercedes@1886
Spidey2025!

                                   [ Read 4 lines ]
^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line
```

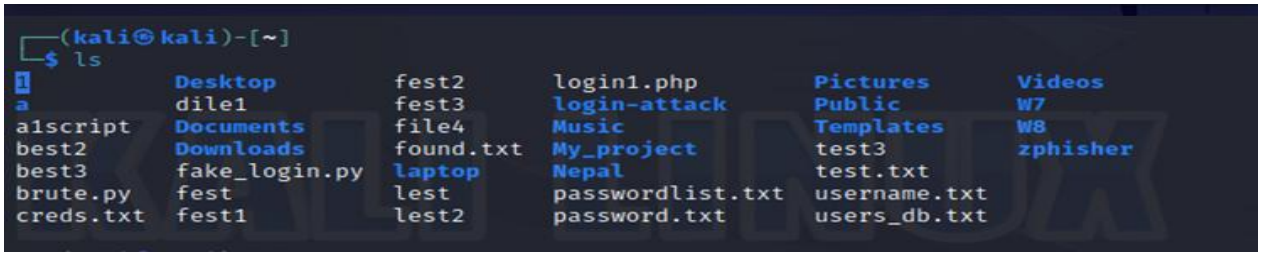


```
┌──(kali㉿kali)-[~]
└─$ cat password.txt
password123
letmein
Mercedes@1886
Spidey2025!

┌──(kali㉿kali)-[~]
└─$
```

Figure 10: Creating list of probable password and checking.

Here you can see files name (fake_login.py, username.txt, password.txt) which was just made.

Command : ls (It lists the files which are available.)

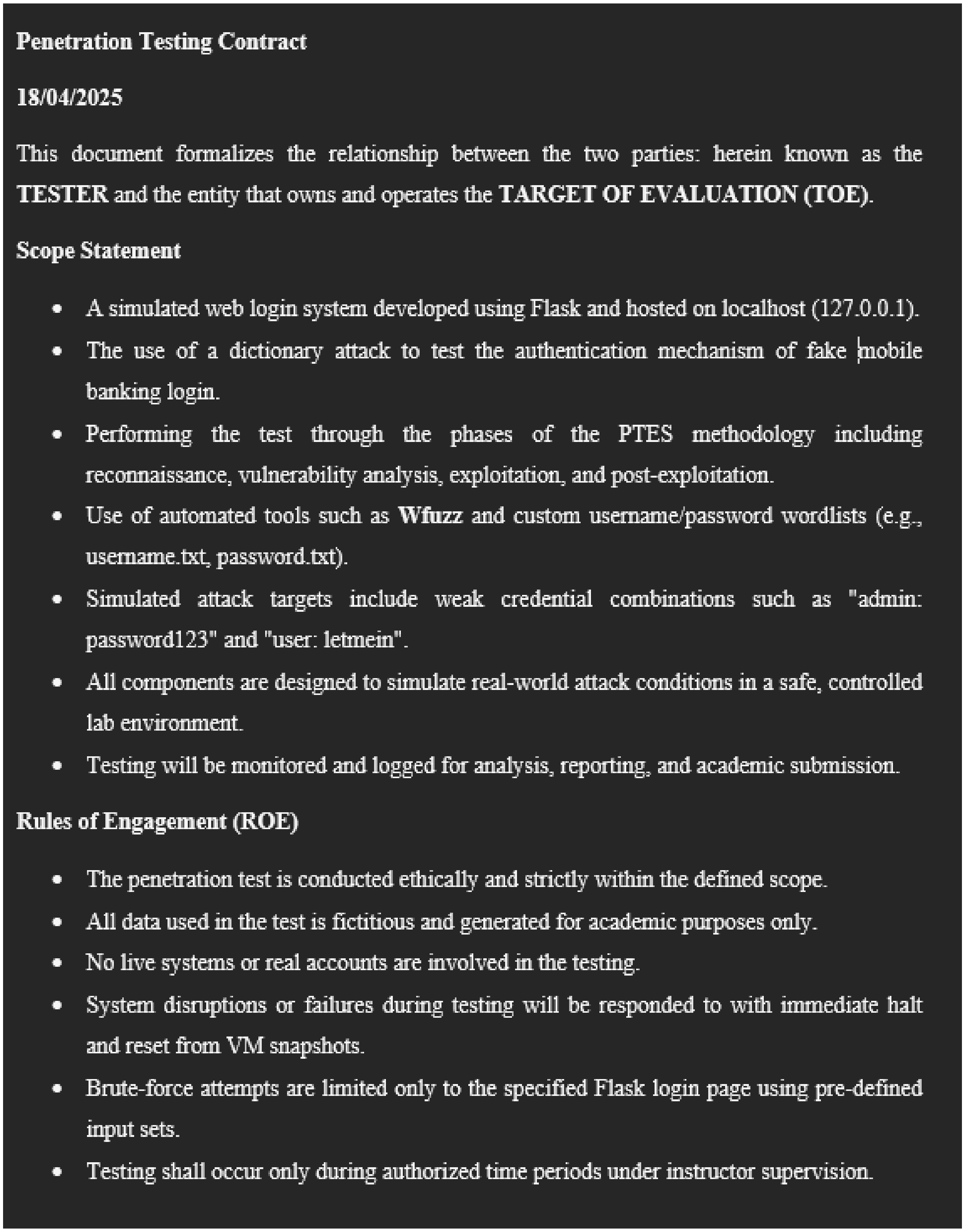


```
┌──(kali㉿kali)-[~]
└─$ ls
1           Desktop          fest2      login1.php        Pictures      Videos
a           dile1            fest3      login-attack      Public        W7
a1script    Documents        file4      Music             Templates     W8
best2       Downloads        found.txt  My_project        test3         zphisher
best3       fake_login.py    laptop     Nepal             test.txt
brute.py    fest             lest       passwordlist.txt  username.txt
creds.txt   fest1            lest2      password.txt      users_db.txt
```

Figure 11: listing all files.

3.1 Pre-engagement Interactions

In this case, this penetration testing contract outlines a scenario that involves simulating an attack against a Flask based web login system to see how strong this system's authentication mechanism is. The test is carried under the guidance of the PTES framework where this test drives dictionary attacks and post authentication access analysis. It lays down strict Rules of Engagement (ROE) as it is no harm to real systems, academic purpose only and use of authorized tools and methods.



Penetration Testing Contract

18/04/2025

This document formalizes the relationship between the two parties: herein known as the **TESTER** and the entity that owns and operates the **TARGET OF EVALUATION (TOE)**.

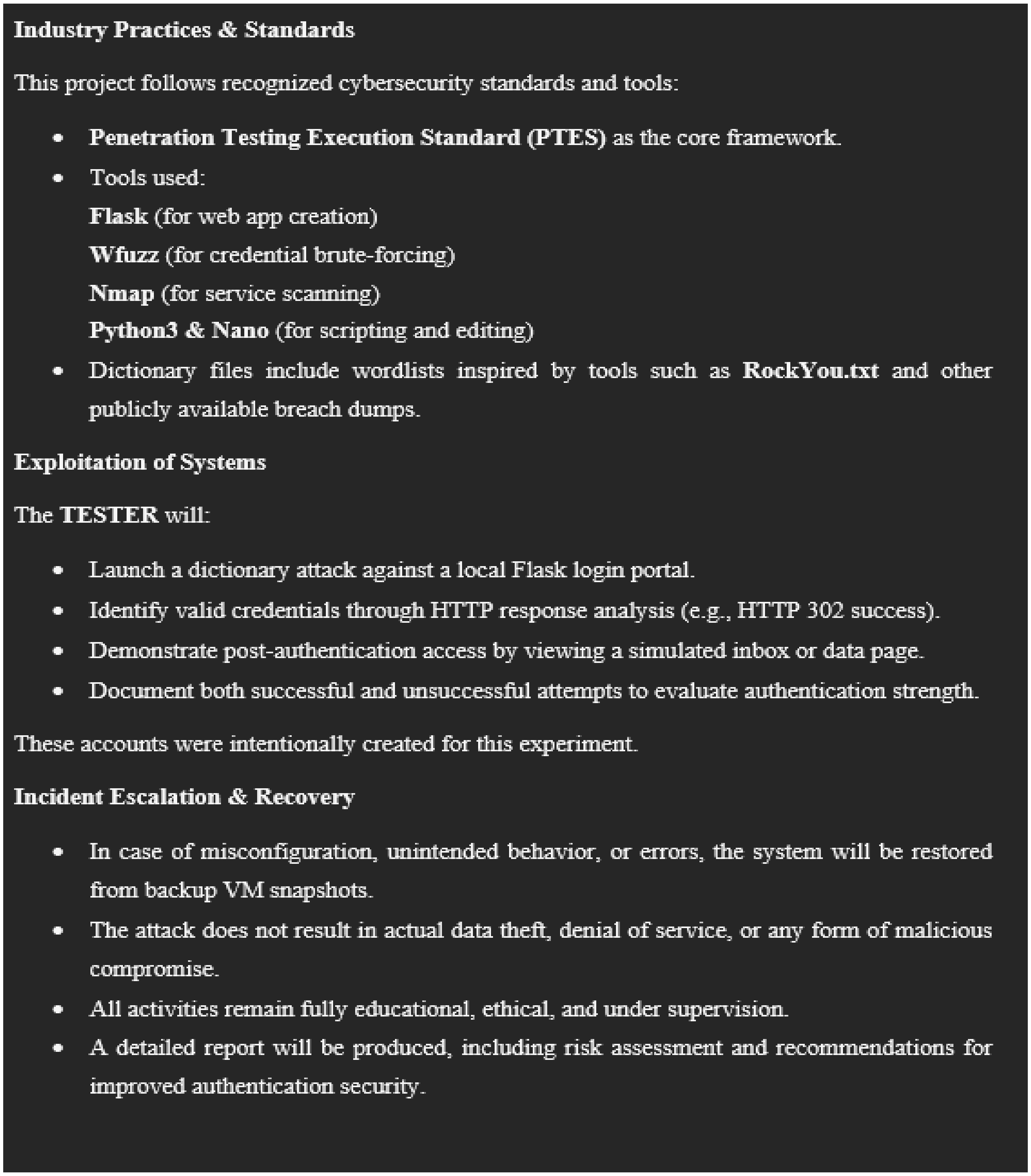
Scope Statement

- A simulated web login system developed using Flask and hosted on localhost (127.0.0.1).
- The use of a dictionary attack to test the authentication mechanism of fake mobile banking login.
- Performing the test through the phases of the PTES methodology including reconnaissance, vulnerability analysis, exploitation, and post-exploitation.
- Use of automated tools such as **Wfuzz** and custom username/password wordlists (e.g., username.txt, password.txt).
- Simulated attack targets include weak credential combinations such as "admin: password123" and "user: letmein".
- All components are designed to simulate real-world attack conditions in a safe, controlled lab environment.
- Testing will be monitored and logged for analysis, reporting, and academic submission.

Rules of Engagement (ROE)

- The penetration test is conducted ethically and strictly within the defined scope.
- All data used in the test is fictitious and generated for academic purposes only.
- No live systems or real accounts are involved in the testing.
- System disruptions or failures during testing will be responded to with immediate halt and reset from VM snapshots.
- Brute-force attempts are limited only to the specified Flask login page using pre-defined input sets.
- Testing shall occur only during authorized time periods under instructor supervision.

Figure 12:Penetration Testing contract pt.1



Industry Practices & Standards

This project follows recognized cybersecurity standards and tools:

- **Penetration Testing Execution Standard (PTES)** as the core framework.
- Tools used:
  Flask (for web app creation)
  Wfuzz (for credential brute-forcing)
  Nmap (for service scanning)
  Python3 & Nano (for scripting and editing)
- Dictionary files include wordlists inspired by tools such as **RockYou.txt** and other publicly available breach dumps.

Exploitation of Systems

The **TESTER** will:

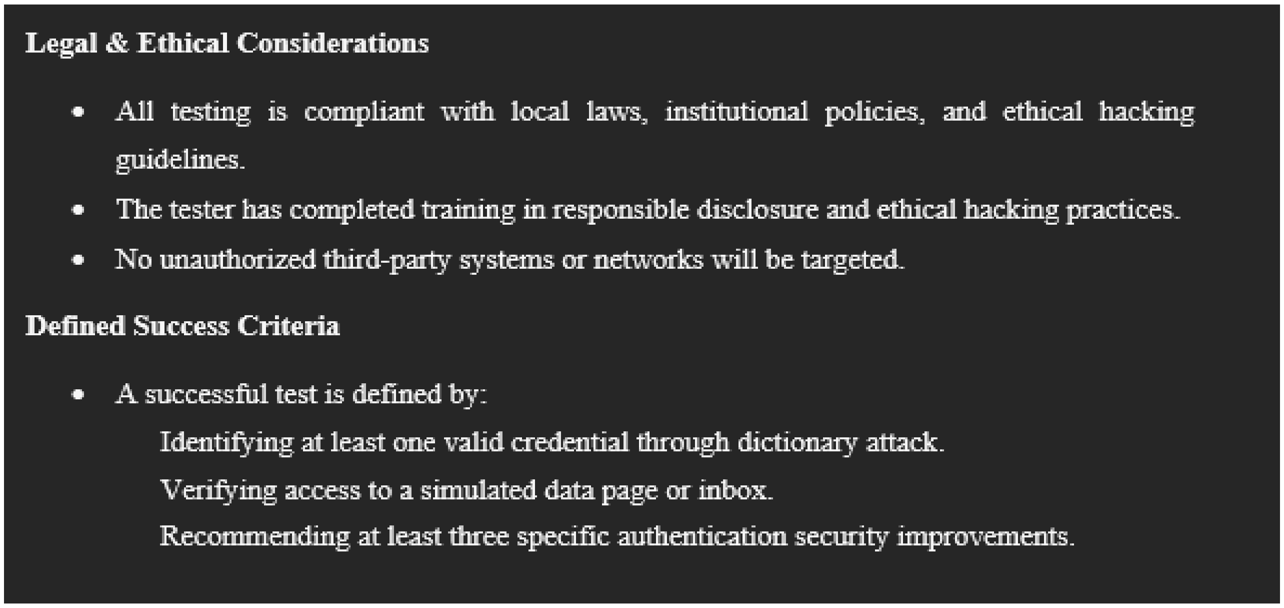
- Launch a dictionary attack against a local Flask login portal.
- Identify valid credentials through HTTP response analysis (e.g., HTTP 302 success).
- Demonstrate post-authentication access by viewing a simulated inbox or data page.
- Document both successful and unsuccessful attempts to evaluate authentication strength.

These accounts were intentionally created for this experiment.

Incident Escalation & Recovery

- In case of misconfiguration, unintended behavior, or errors, the system will be restored from backup VM snapshots.
- The attack does not result in actual data theft, denial of service, or any form of malicious compromise.
- All activities remain fully educational, ethical, and under supervision.
- A detailed report will be produced, including risk assessment and recommendations for improved authentication security.

Figure 13: Penetration Testing contract pt.2



Legal & Ethical Considerations

- All testing is compliant with local laws, institutional policies, and ethical hacking guidelines.
- The tester has completed training in responsible disclosure and ethical hacking practices.
- No unauthorized third-party systems or networks will be targeted.

Defined Success Criteria

- A successful test is defined by:
  Identifying at least one valid credential through dictionary attack.
  Verifying access to a simulated data page or inbox.
  Recommending at least three specific authentication security improvements.

Figure 14: Penetration Testing contract pt.3

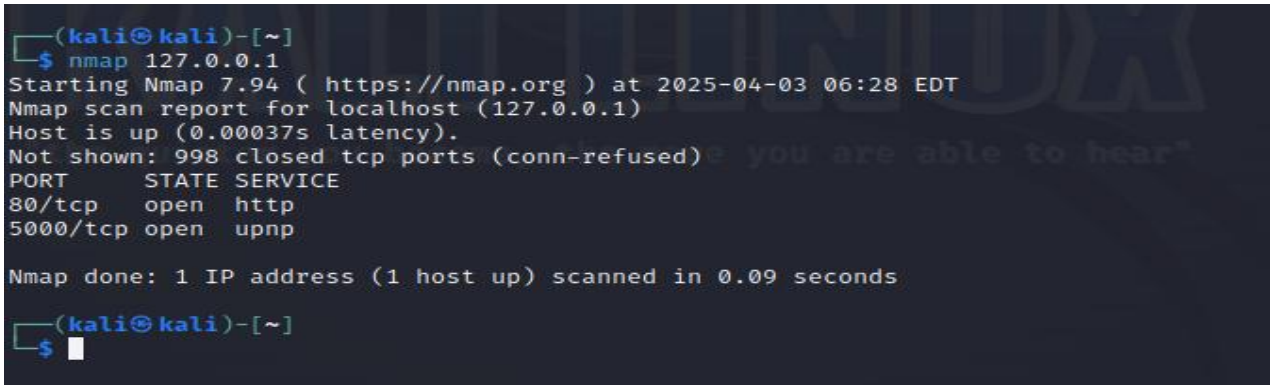
3.2 Information Gathering

In this phase, we are gathering the data needed for an attack.

Step 6: Using scanning tool (nmap).

Command is **nmap 127.0.0.1** This command performs a simple TCP scan on localhost. It checks the most common 1000 ports on (127.0.0.1). It found two open ports:

80/tcp: HTTP (web server), 5000/tcp: UPNP (Universal Plug and Play)

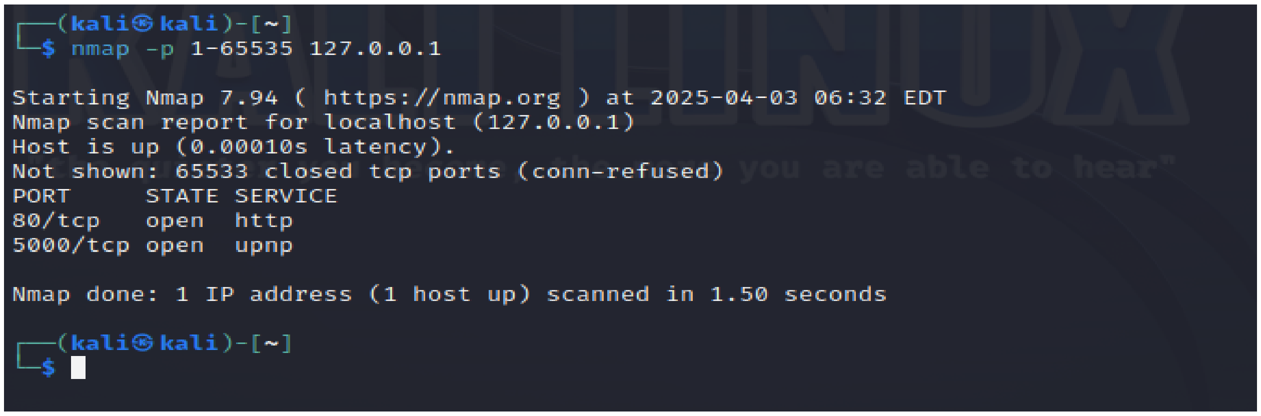


```
┌──(kali㉿kali)-[~]
└─$ nmap 127.0.0.1
Starting Nmap 7.94 ( https://nmap.org ) at 2025-04-03 06:28 EDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00037s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT     STATE SERVICE
80/tcp   open  http
5000/tcp open  upnp

Nmap done: 1 IP address (1 host up) scanned in 0.09 seconds

┌──(kali㉿kali)-[~]
└─$ 
```

Figure 15: scanning ip address.

Command is **nmap –p 1-65535 127.0.0.1**

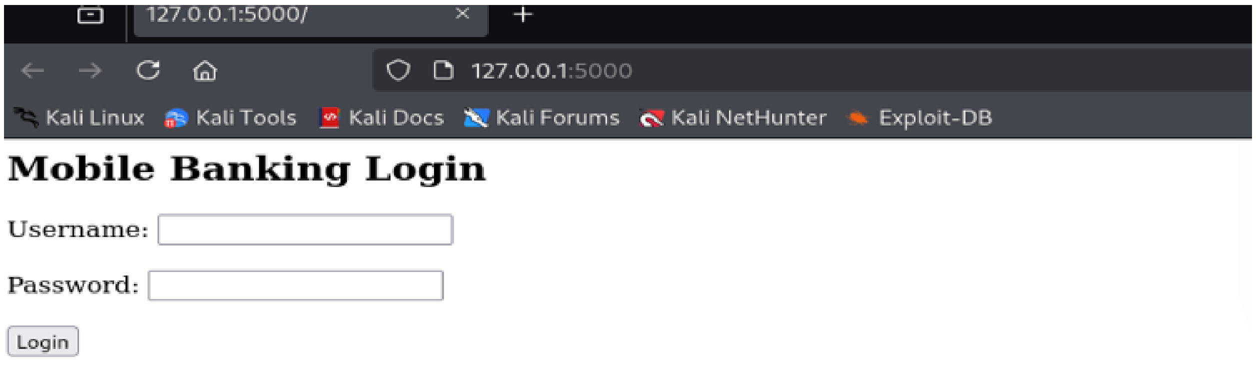
Nmap scans all 65,535 TCP ports instead of just the top 1000. It still reports ports 80 and 5000 as open, confirming the presence of HTTP and UPNP services.



```
┌──(kali㉿kali)-[~]
└─$ nmap -p 1-65535 127.0.0.1

Starting Nmap 7.94 ( https://nmap.org ) at 2025-04-03 06:32 EDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00010s latency).
Not shown: 65533 closed tcp ports (conn-refused)
PORT     STATE SERVICE
80/tcp   open  http
5000/tcp open  upnp

Nmap done: 1 IP address (1 host up) scanned in 1.50 seconds

┌──(kali㉿kali)-[~]
└─$ 
```

3.3 Threat modelling

The page does not provide simple security measures such as CAPTCHA, MFA (Multi Factor Authentication) and account lockout protection, increasing the risk of an attacker to try automated login.



Mobile Banking Login

Username:

Password:

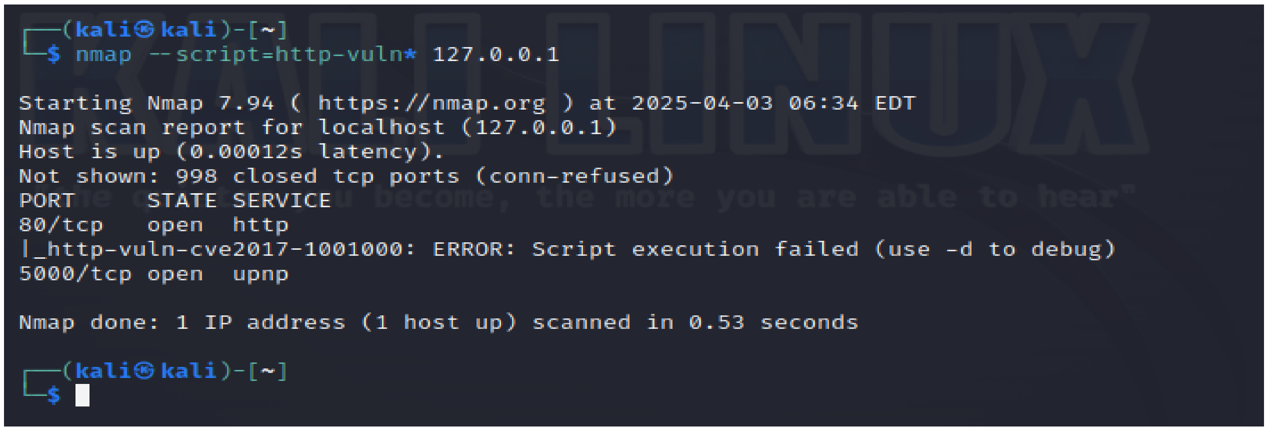
Login

3.4 Vulnerability Analysis

This phase identifies weaknesses in the login system that could be exploited using the gathered credentials.

Step 7: Using Nmap to run vulnerability scripts or version scans to look for known exploits.

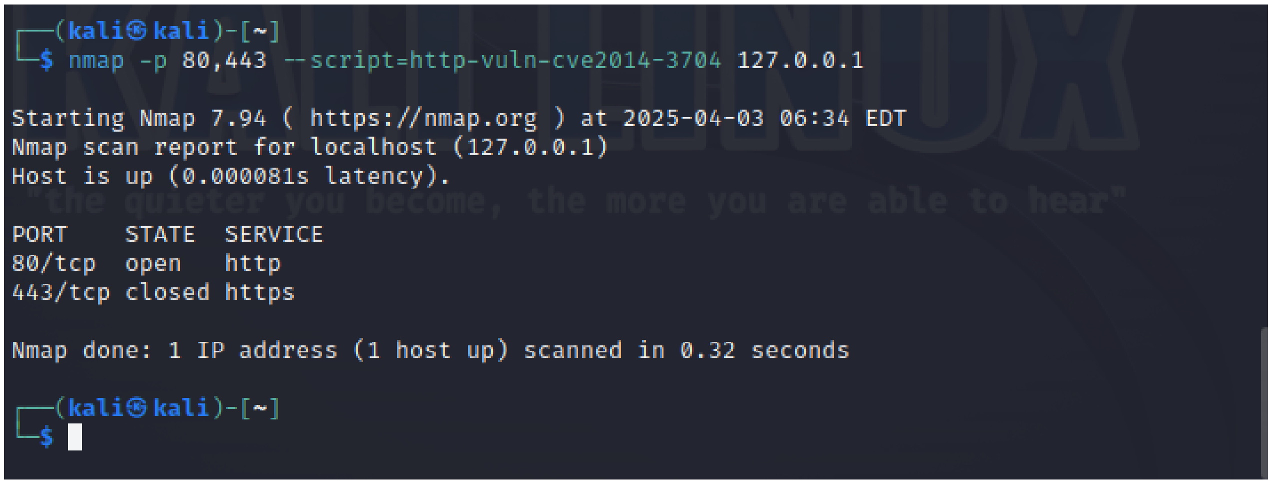
Command is **nmap --script=http-vuln* 127.0.0.1** This command runs all vulnerability scanning scripts that begin with http-vuln related to HTTP services. But the script fails, which indicate the service is not vulnerable or does not respond as expected.



```
┌──(kali㉿kali)-[~]
└─$ nmap --script=http-vuln* 127.0.0.1

Starting Nmap 7.94 ( https://nmap.org ) at 2025-04-03 06:34 EDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00012s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT     STATE SERVICE
80/tcp   open  http
|_http-vuln-cve2017-1001000: ERROR: Script execution failed (use -d to debug)
5000/tcp open  upnp

Nmap done: 1 IP address (1 host up) scanned in 0.53 seconds

┌──(kali㉿kali)-[~]
└─$ 
```

Command is **nmap -p 80,443 --script=http-vuln-cve2014-3704 127.0.0.1**

This scan targets HTTP and HTTPS ports to check for a specific known vulnerability in Drupal (CVE-2014-3704). Port 443 is closed, and no vulnerability was reported on port 80. Nmap can be used not only to scan ports but also to identify potential security risks.



```
┌──(kali㉿kali)-[~]
└─$ nmap -p 80,443 --script=http-vuln-cve2014-3704 127.0.0.1

Starting Nmap 7.94 ( https://nmap.org ) at 2025-04-03 06:34 EDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000081s latency).

PORT    STATE  SERVICE
80/tcp  open   http
443/tcp closed https

Nmap done: 1 IP address (1 host up) scanned in 0.32 seconds

┌──(kali㉿kali)-[~]
└─$ 
```

Figure 16: Using Nmap to run vulnerability scripts or version scans to look for known exploits.

3.5 Exploitation

The crafted wordlists is used to launch a dictionary attack and attempt to gain access to the system.

Step 8: Using Wfuzz to perform a brute-force login attack.

Command is **wfuzz -z file,username.txt -z file,password.txt**

**-d "username=FUZZ&password=FUZZ2" **

--hc=403 http://127.0.0.1:5000/login

-z file,username.txt: Use username.txt as the first wordlist.

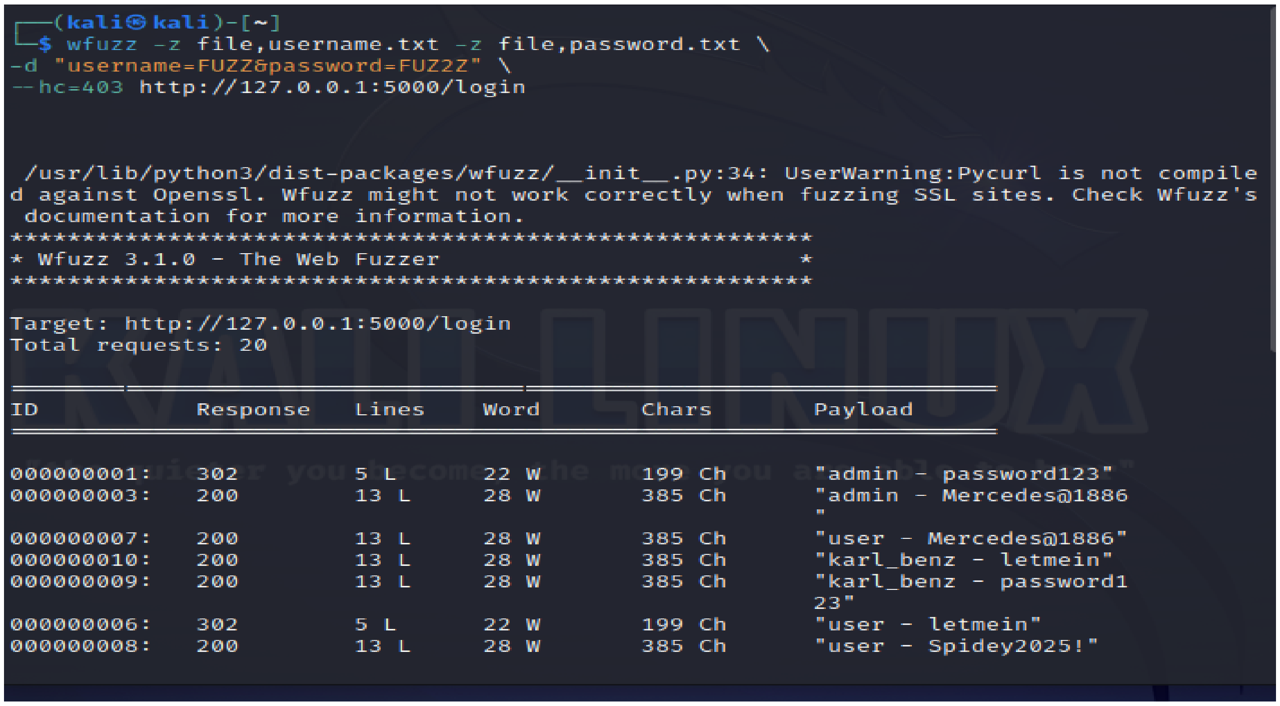
-z file,password.txt: Use password.txt as the second wordlist.

-d "username=FUZZ&password=FUZZ2": Substitutes FUZZ with usernames and FUZZ2 with passwords.

--hc=403: Hide responses with HTTP code 403 (Forbidden), as they indicate failure.

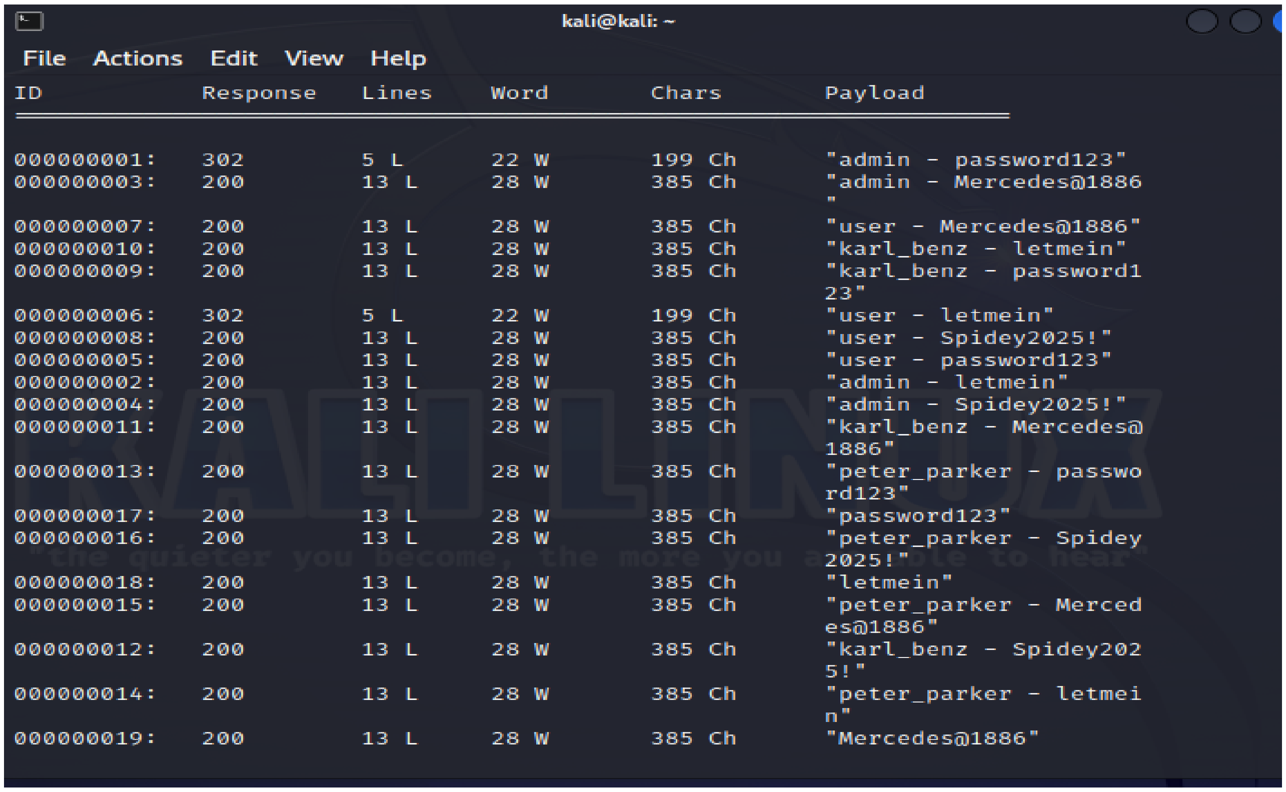
http://127.0.0.1:5000/login: The login endpoint which we are attacking.

In wfuzz, to check which is correct it should be reviewed timely. If there is 200 it means, it is incorrect and if it is 302 it means, it is correct.

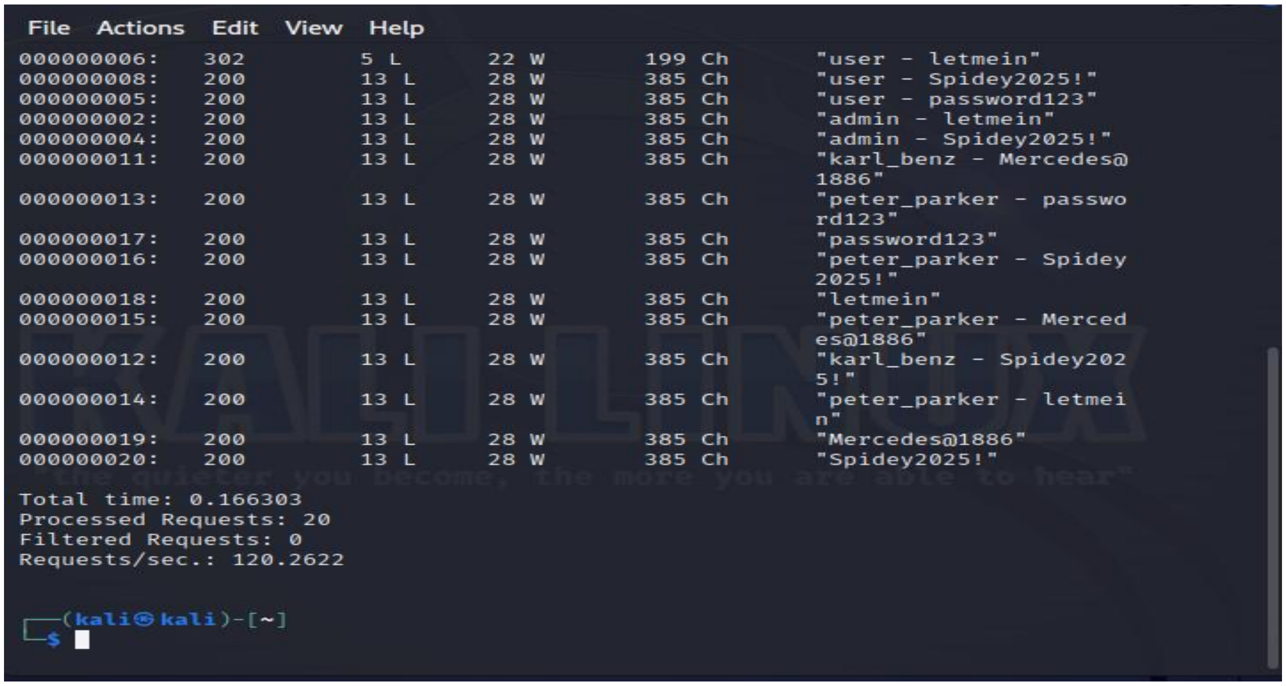


```
┌──(kali㉿kali)-[~]
└─$ wfuzz -z file,username.txt -z file,password.txt \
-d "username=FUZZ&password=FUZ2Z" \
--hc=403 http://127.0.0.1:5000/login


 /usr/lib/python3/dist-packages/wfuzz/__init__.py:34: UserWarning:Pycurl is not compile
d against Openssl. Wfuzz might not work correctly when fuzzing SSL sites. Check Wfuzz's
 documentation for more information.
********************************************************
* Wfuzz 3.1.0 - The Web Fuzzer                         *
********************************************************

Target: http://127.0.0.1:5000/login
Total requests: 20

=====================================================================
ID           Response   Lines    Word       Chars       Payload
=====================================================================

000000001:   302        5 L      22 W       199 Ch      "admin - password123"
000000003:   200        13 L     28 W       385 Ch      "admin - Mercedes@1886
                                                        "
000000007:   200        13 L     28 W       385 Ch      "user - Mercedes@1886"
000000010:   200        13 L     28 W       385 Ch      "karl_benz - letmein"
000000009:   200        13 L     28 W       385 Ch      "karl_benz - password1
                                                        23"
000000006:   302        5 L      22 W       199 Ch      "user - letmein"
000000008:   200        13 L     28 W       385 Ch      "user - Spidey2025!"
```

Figure 17: Using wfuzz to perform brute-force login attack.

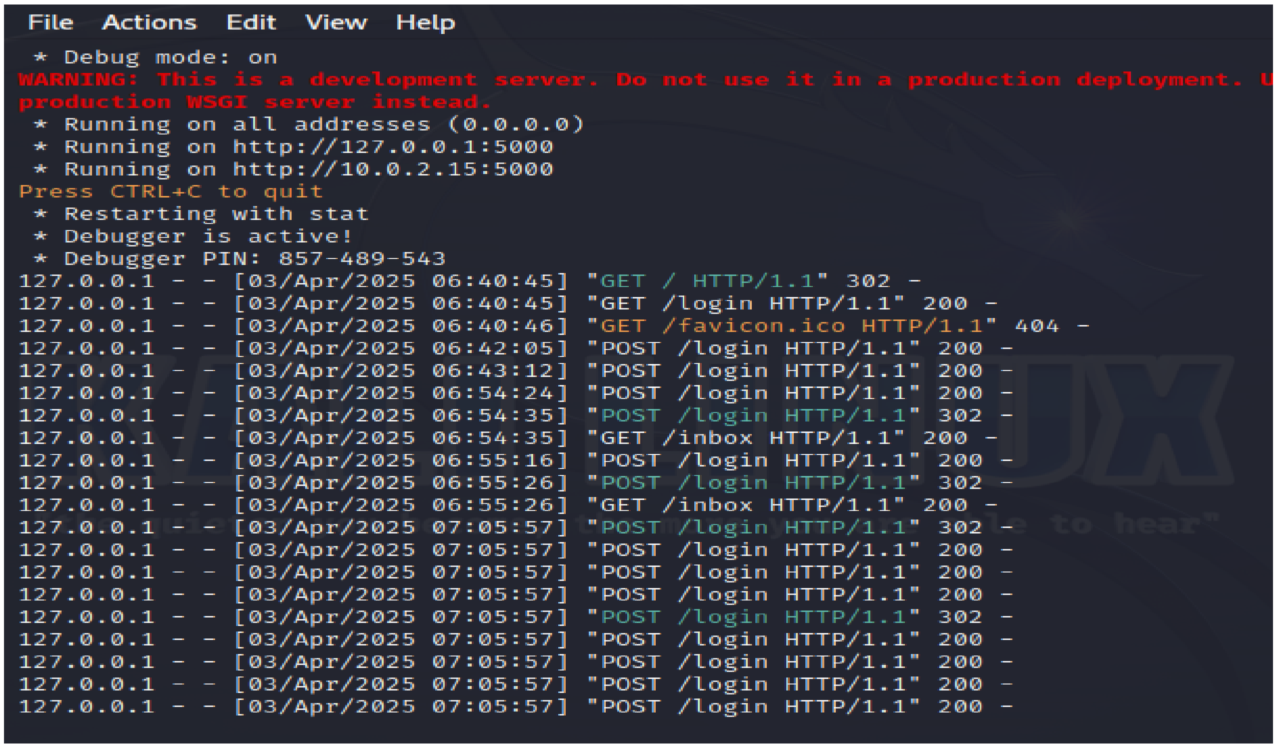


```
ID              Response    Lines     Word        Chars        Payload
=========================================================================

000000001:      302         5 L       22 W        199 Ch       "admin - password123"
000000003:      200         13 L      28 W        385 Ch       "admin - Mercedes@1886
                                                               "
000000007:      200         13 L      28 W        385 Ch       "user - Mercedes@1886"
000000010:      200         13 L      28 W        385 Ch       "karl_benz - letmein"
000000009:      200         13 L      28 W        385 Ch       "karl_benz - password1
                                                               23"
000000006:      302         5 L       22 W        199 Ch       "user - letmein"
000000008:      200         13 L      28 W        385 Ch       "user - Spidey2025!"
000000005:      200         13 L      28 W        385 Ch       "user - password123"
000000002:      200         13 L      28 W        385 Ch       "admin - letmein"
000000004:      200         13 L      28 W        385 Ch       "admin - Spidey2025!"
000000011:      200         13 L      28 W        385 Ch       "karl_benz - Mercedes@
                                                               1886"
000000013:      200         13 L      28 W        385 Ch       "peter_parker - passwo
                                                               rd123"
000000017:      200         13 L      28 W        385 Ch       "password123"
000000016:      200         13 L      28 W        385 Ch       "peter_parker - Spidey
                                                               2025!"
000000018:      200         13 L      28 W        385 Ch       "letmein"
000000015:      200         13 L      28 W        385 Ch       "peter_parker - Merced
                                                               es@1886"
000000012:      200         13 L      28 W        385 Ch       "karl_benz - Spidey202
                                                               5!"
000000014:      200         13 L      28 W        385 Ch       "peter_parker - letmei
                                                               n"
000000019:      200         13 L      28 W        385 Ch       "Mercedes@1886"
```

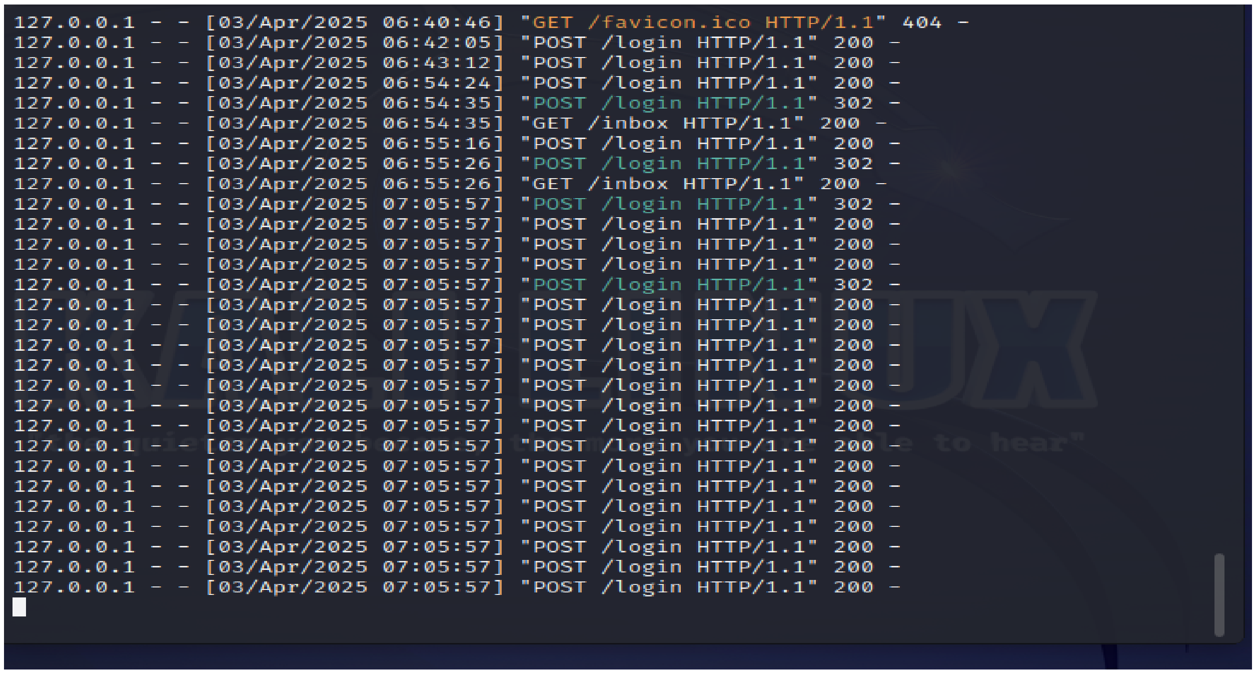


```
000000006:      302         5 L       22 W        199 Ch       "user - letmein"
000000008:      200         13 L      28 W        385 Ch       "user - Spidey2025!"
000000005:      200         13 L      28 W        385 Ch       "user - password123"
000000002:      200         13 L      28 W        385 Ch       "admin - letmein"
000000004:      200         13 L      28 W        385 Ch       "admin - Spidey2025!"
000000011:      200         13 L      28 W        385 Ch       "karl_benz - Mercedes@
                                                               1886"
000000013:      200         13 L      28 W        385 Ch       "peter_parker - passwo
                                                               rd123"
000000017:      200         13 L      28 W        385 Ch       "password123"
000000016:      200         13 L      28 W        385 Ch       "peter_parker - Spidey
                                                               2025!"
000000018:      200         13 L      28 W        385 Ch       "letmein"
000000015:      200         13 L      28 W        385 Ch       "peter_parker - Merced
                                                               es@1886"
000000012:      200         13 L      28 W        385 Ch       "karl_benz - Spidey202
                                                               5!"
000000014:      200         13 L      28 W        385 Ch       "peter_parker - letmei
                                                               n"
000000019:      200         13 L      28 W        385 Ch       "Mercedes@1886"
000000020:      200         13 L      28 W        385 Ch       "Spidey2025!"

Total time: 0.166303
Processed Requests: 20
Filtered Requests: 0
Requests/sec.: 120.2622

┌──(kali㉿kali)-[~]
└─$ █
```

In step 3, python3 was used to run Python file. In python3 the attacks attempts are displayed.



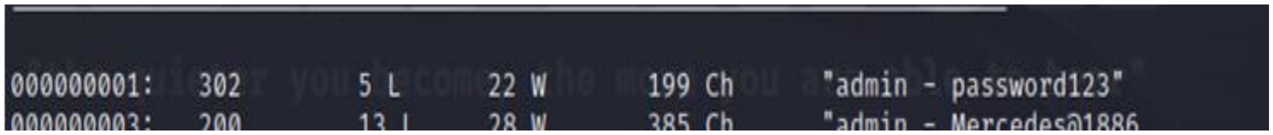
```
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. U
production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://10.0.2.15:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 857-489-543
127.0.0.1 - - [03/Apr/2025 06:40:45] "GET / HTTP/1.1" 302 -
127.0.0.1 - - [03/Apr/2025 06:40:45] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 06:40:46] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [03/Apr/2025 06:42:05] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 06:43:12] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 06:54:24] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 06:54:35] "POST /login HTTP/1.1" 302 -
127.0.0.1 - - [03/Apr/2025 06:54:35] "GET /inbox HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 06:55:16] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 06:55:26] "POST /login HTTP/1.1" 302 -
127.0.0.1 - - [03/Apr/2025 06:55:26] "GET /inbox HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 302 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 302 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
```



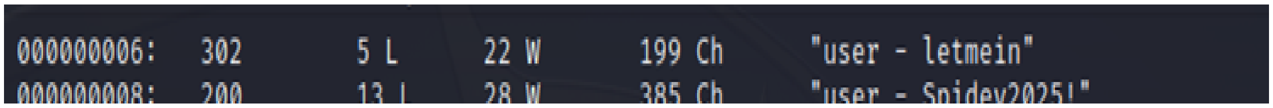
```
127.0.0.1 - - [03/Apr/2025 06:40:46] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [03/Apr/2025 06:42:05] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 06:43:12] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 06:54:24] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 06:54:35] "POST /login HTTP/1.1" 302 -
127.0.0.1 - - [03/Apr/2025 06:54:35] "GET /inbox HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 06:55:16] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 06:55:26] "POST /login HTTP/1.1" 302 -
127.0.0.1 - - [03/Apr/2025 06:55:26] "GET /inbox HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 302 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 302 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2025 07:05:57] "POST /login HTTP/1.1" 200 -
```

So, these are the correct data.

“admin – password123” and **“user - letmein”** two correct username and passwords.



```
000000001:   302        5 L      22 W      199 Ch      "admin - password123"
000000003:   200        13 L     28 W      385 Ch      "admin - Mercedes@1886
```



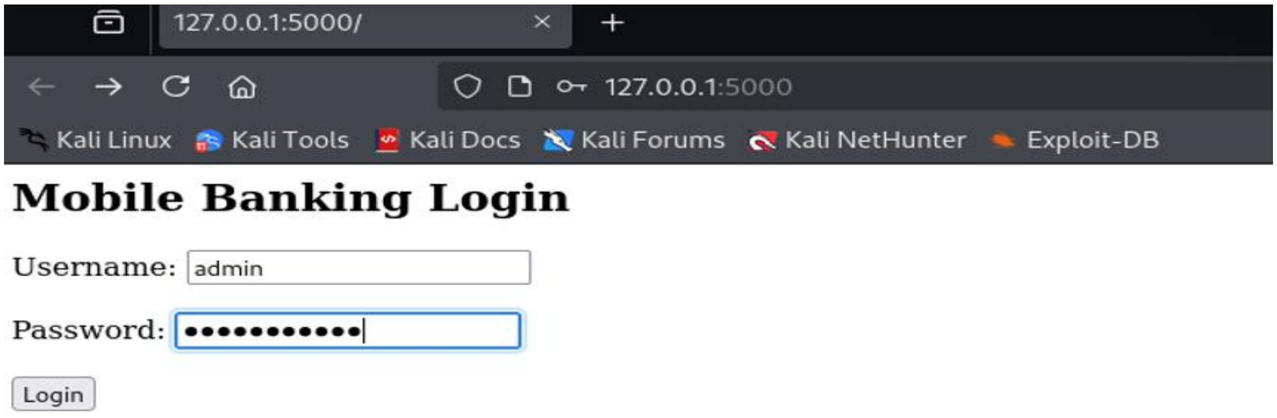
```
000000006:   302        5 L      22 W      199 Ch      "user - letmein"
000000008:   200        13 L     28 W      385 Ch      "user - Spidey2025!"
```

Figure 18: Displaying correct data.

3.6 Post-Exploitation

Step 9: Testing.

Using "admin – password123".



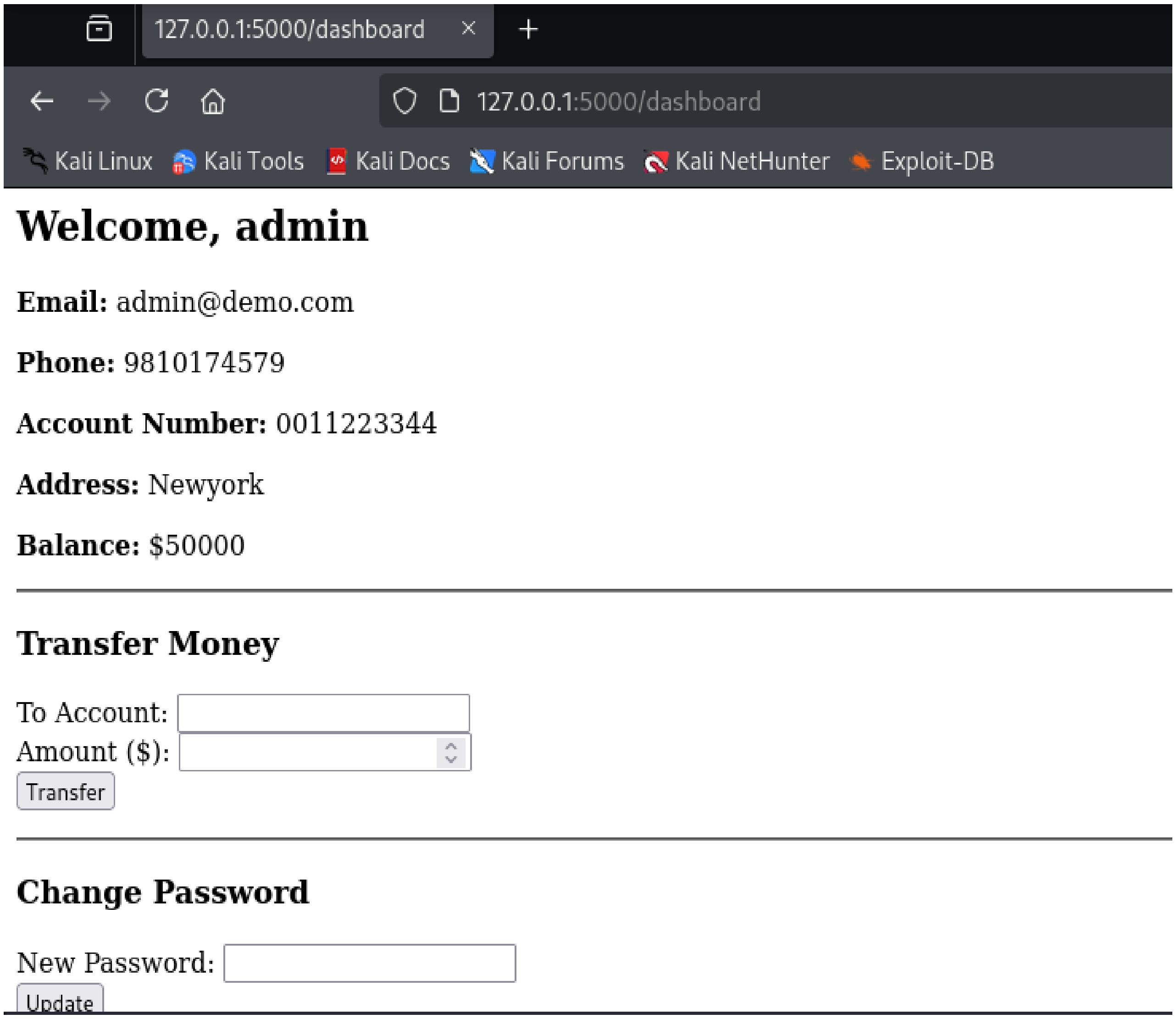
127.0.0.1:5000/

Mobile Banking Login

Username: admin

Password: ●●●●●●●●●●●

The account is open now using the correct data which is just exploit.


127.0.0.1:5000/dashboard

Welcome, admin

Email: admin@demo.com

Phone: 9810174579

Account Number: 0011223344

Address: Newyork

Balance: $50000

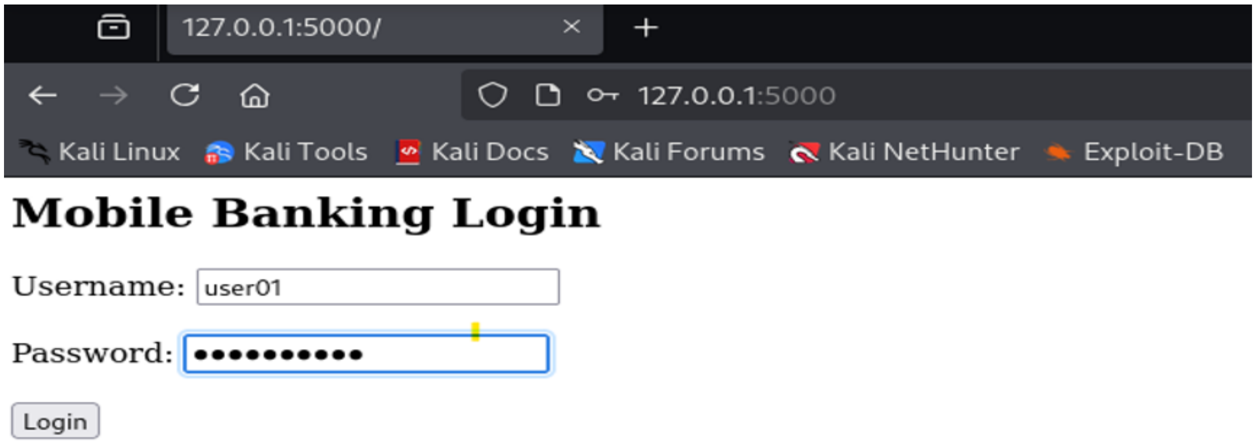
Transfer Money

To Account:
Amount ($):
Transfer

Change Password

New Password:
Update

Using “user - letmein”



127.0.0.1:5000/

127.0.0.1:5000

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB

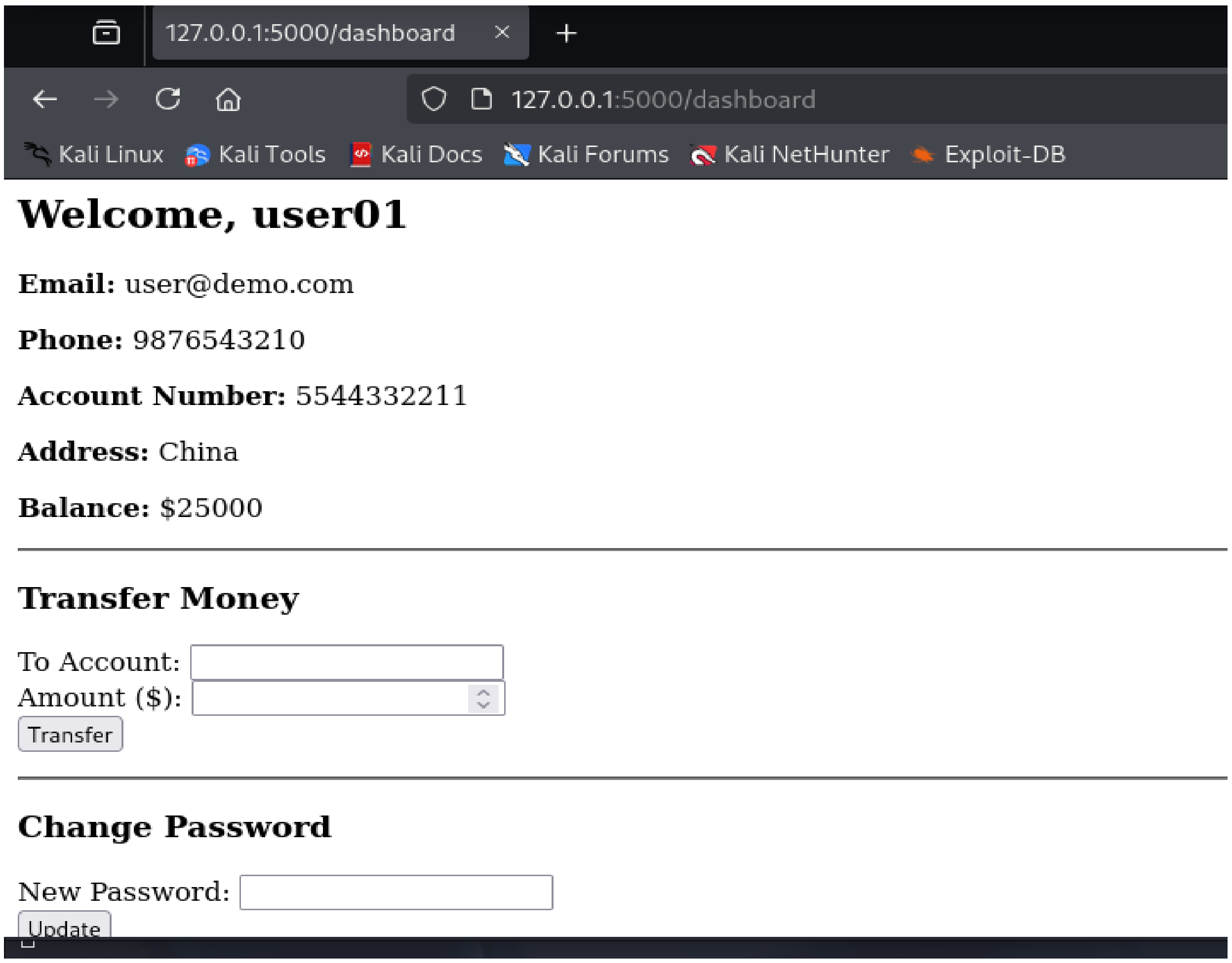
Mobile Banking Login

Username: user01

Password: ●●●●●●●●●●●

Login

The account is open now using the correct data which is just exploit.



127.0.0.1:5000/dashboard

127.0.0.1:5000/dashboard

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB

Welcome, user01

Email: user@demo.com

Phone: 9876543210

Account Number: 5544332211

Address: China

Balance: $25000

Transfer Money

To Account:

Amount ($):

Transfer

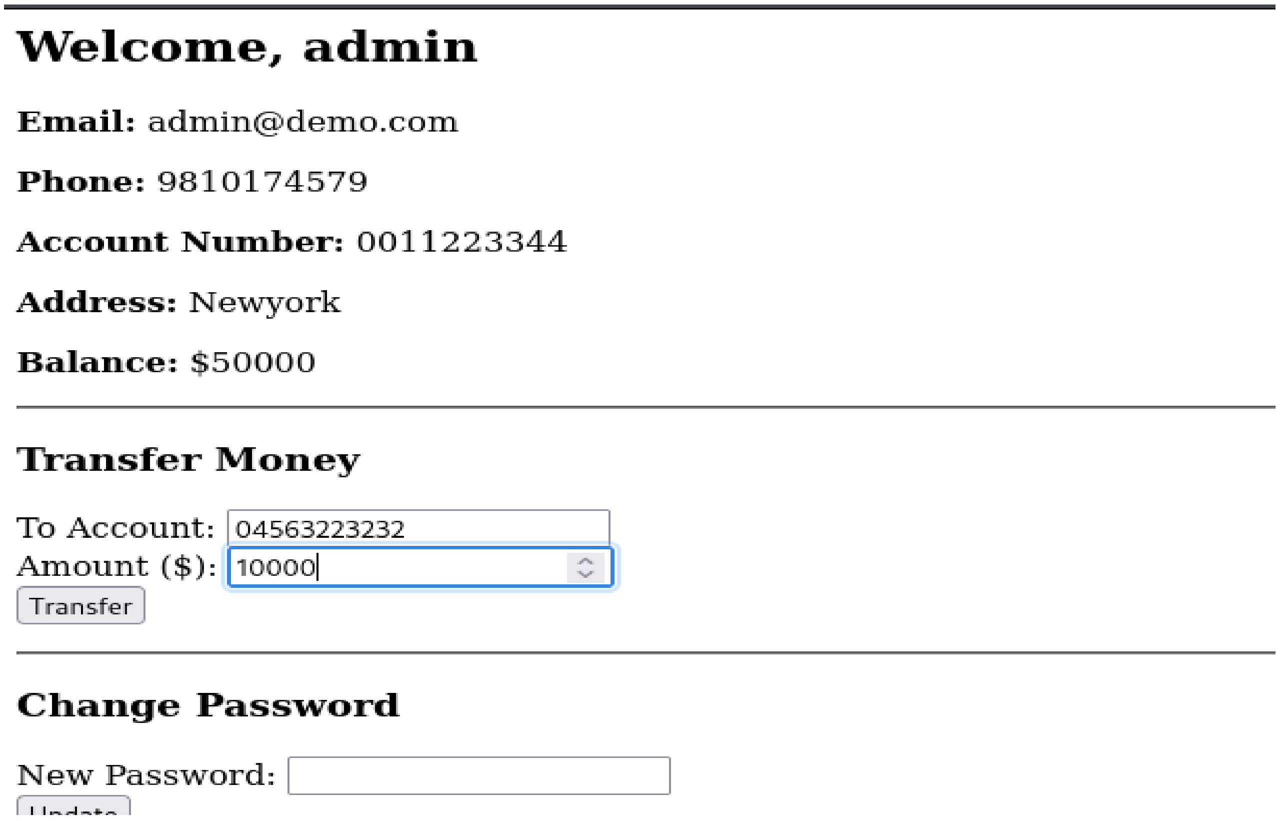
Change Password

New Password:

Update

Both attempts work properly. These are the data which is now exploited. After the attack this page opens, this is bank accounts of an individual. Attackers can use this account for transfer money with access and can change passwords.

Using **“admin – password123”.**



Welcome, admin

Email: admin@demo.com

Phone: 9810174579

Account Number: 0011223344

Address: Newyork

Balance: $50000

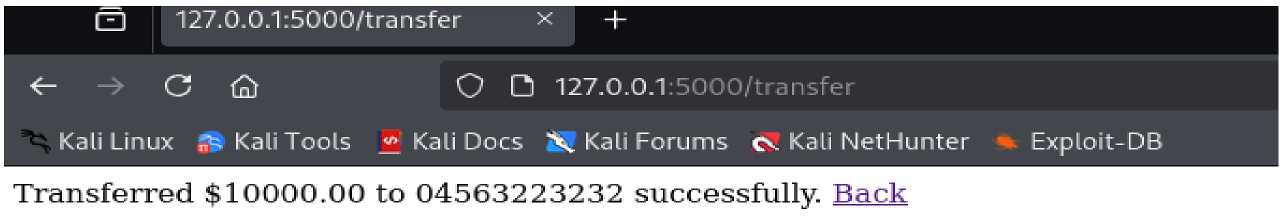
Transfer Money

To Account: 04563223232
Amount ($): 10000
Transfer

Change Password

New Password:

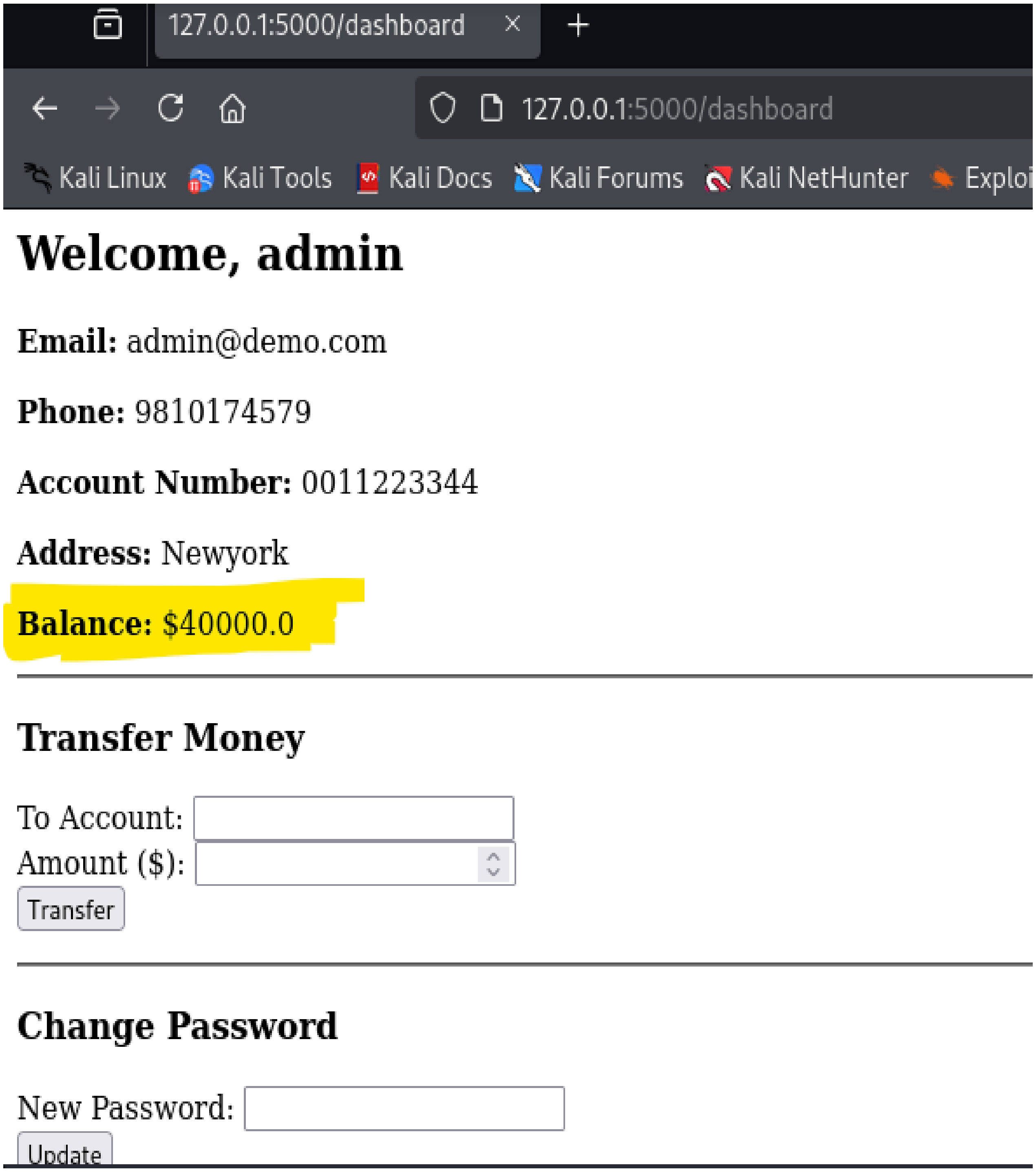
Figure 19:Trying to transfer money to other account.



127.0.0.1:5000/transfer

Transferred $10000.00 to 04563223232 successfully. Back

Figure 20:Successfully transferred.



127.0.0.1:5000/dashboard

Welcome, admin

Email: admin@demo.com

Phone: 9810174579

Account Number: 0011223344

Address: Newyork

Balance: $40000.0

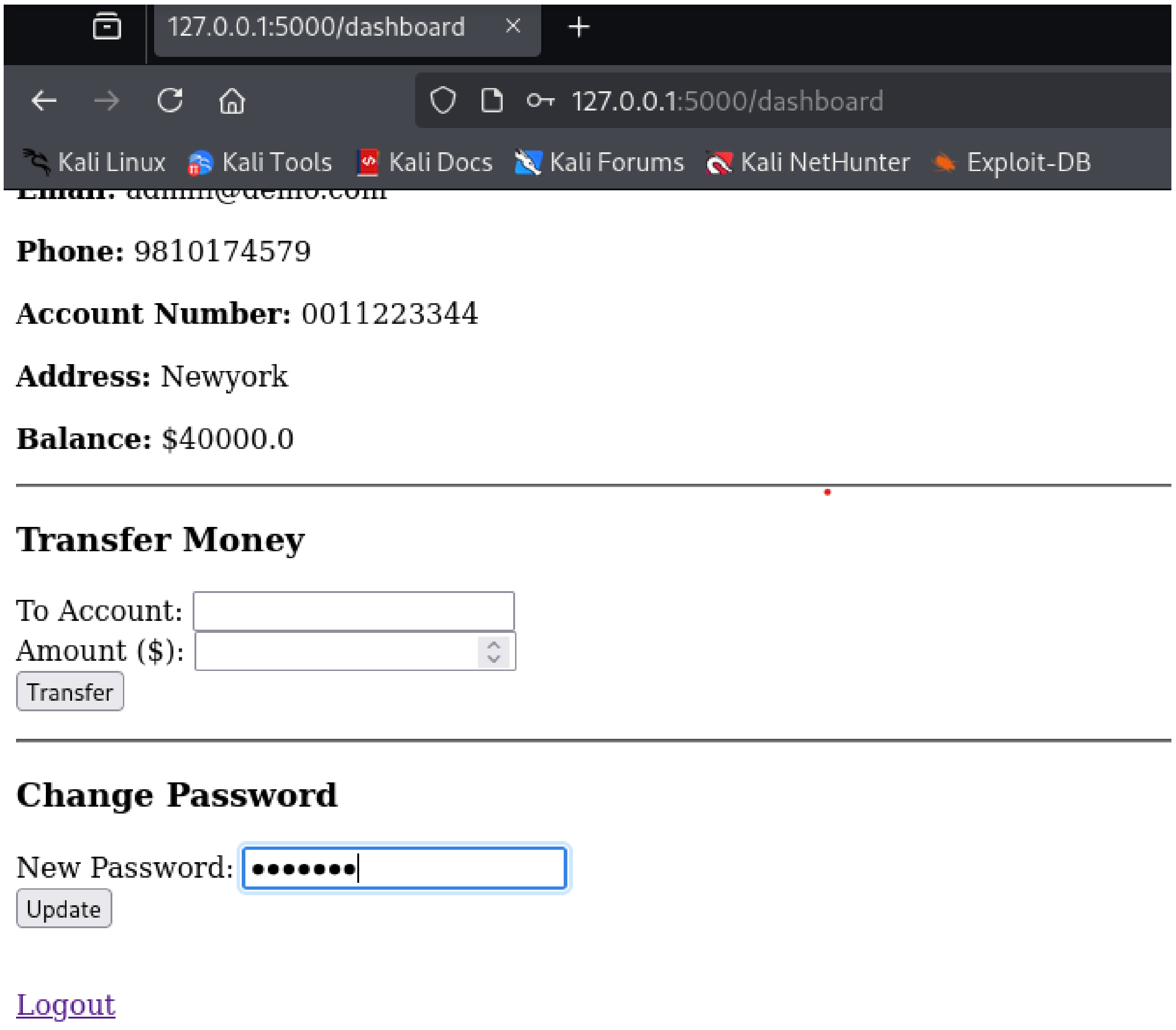
Transfer Money

To Account:	
Amount ($):	

Change Password

New Password:	

Figure 21:The amount is decreased after sending to other account.



127.0.0.1:5000/dashboard

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB

Phone: 9810174579

Account Number: 0011223344

Address: Newyork

Balance: $40000.0

Transfer Money

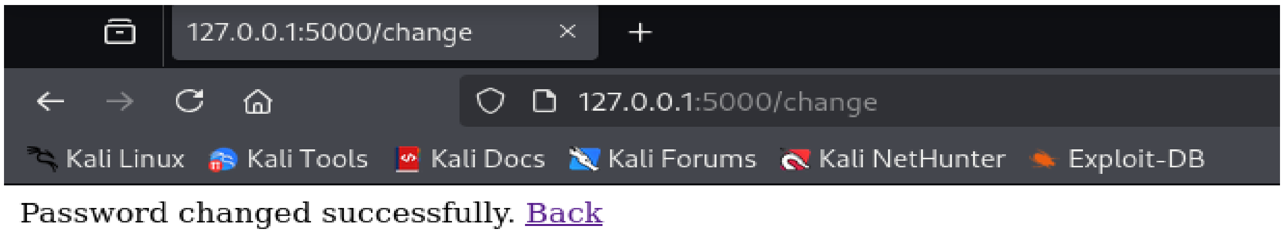
To Account:
Amount ($):
Transfer

Change Password

New Password: ••••••••
Update

Logout

Figure 22:Changing password of the account.

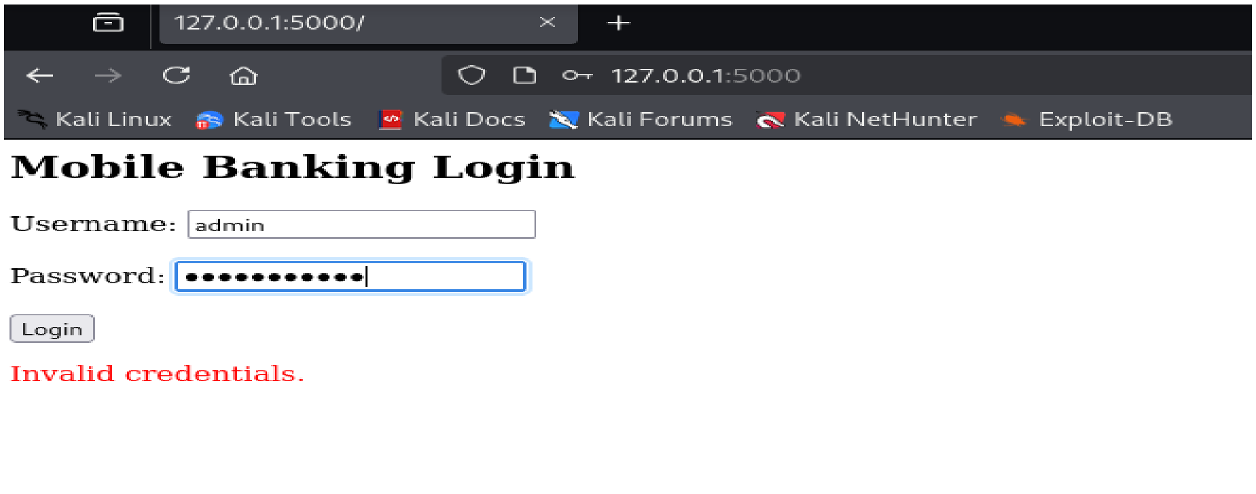


127.0.0.1:5000/change

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB

Password changed successfully. Back

Figure 23:Successfully changed.


127.0.0.1:5000/

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB

Mobile Banking Login

Username: admin

Password: ●●●●●●●●●●●●

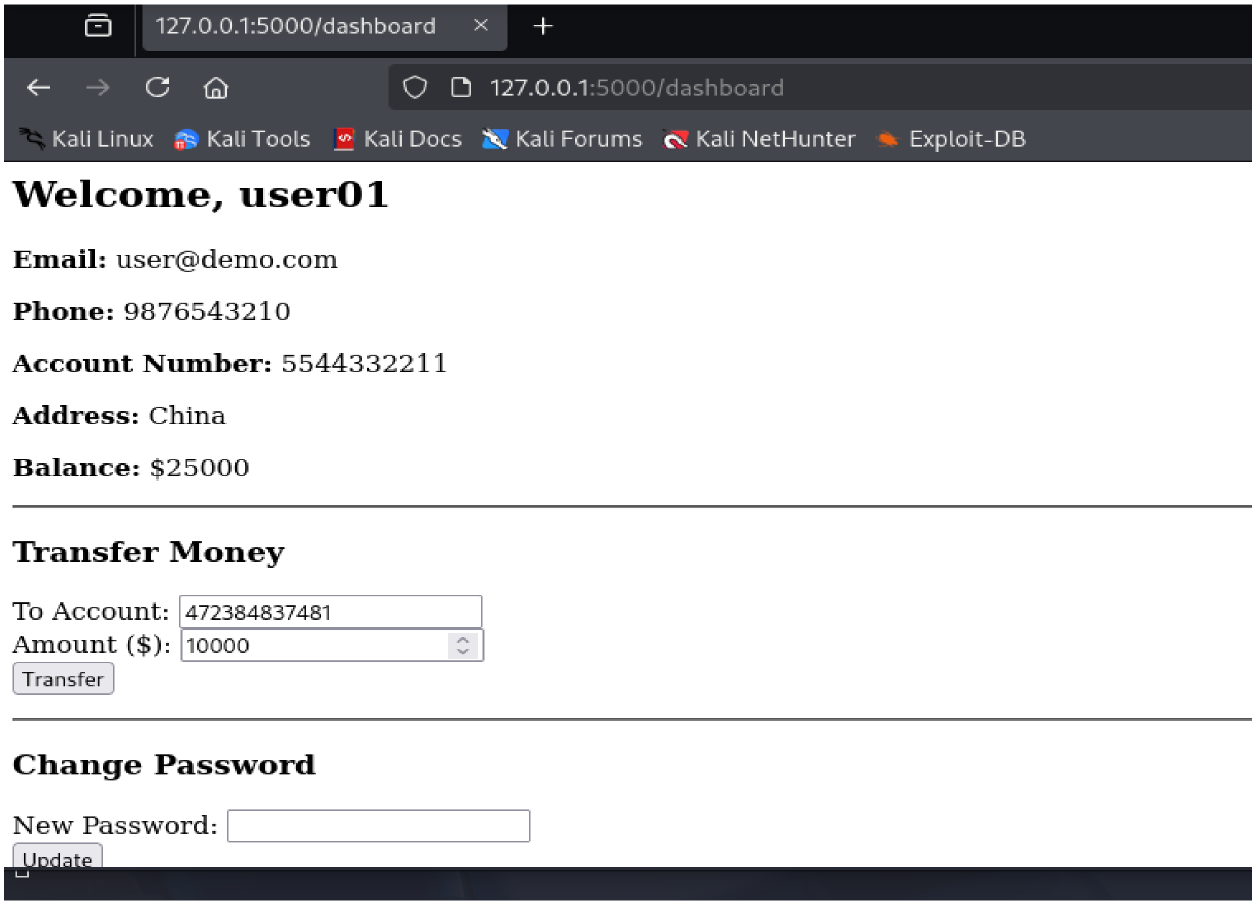
Login

Invalid credentials.

Figure 24:Trying previous credentials.

It doesn't work now, cause the password is already changed.

Using **"user - letmein"**


127.0.0.1:5000/dashboard

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB

Welcome, user01

Email: user@demo.com

Phone: 9876543210

Account Number: 5544332211

Address: China

Balance: $25000

Transfer Money

To Account: 472384837481

Amount ($): 10000

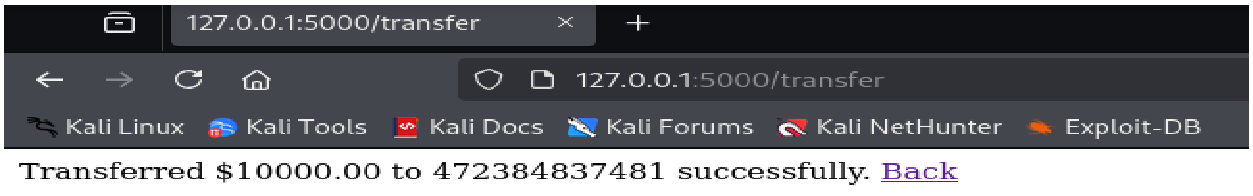
Transfer

Change Password

New Password:

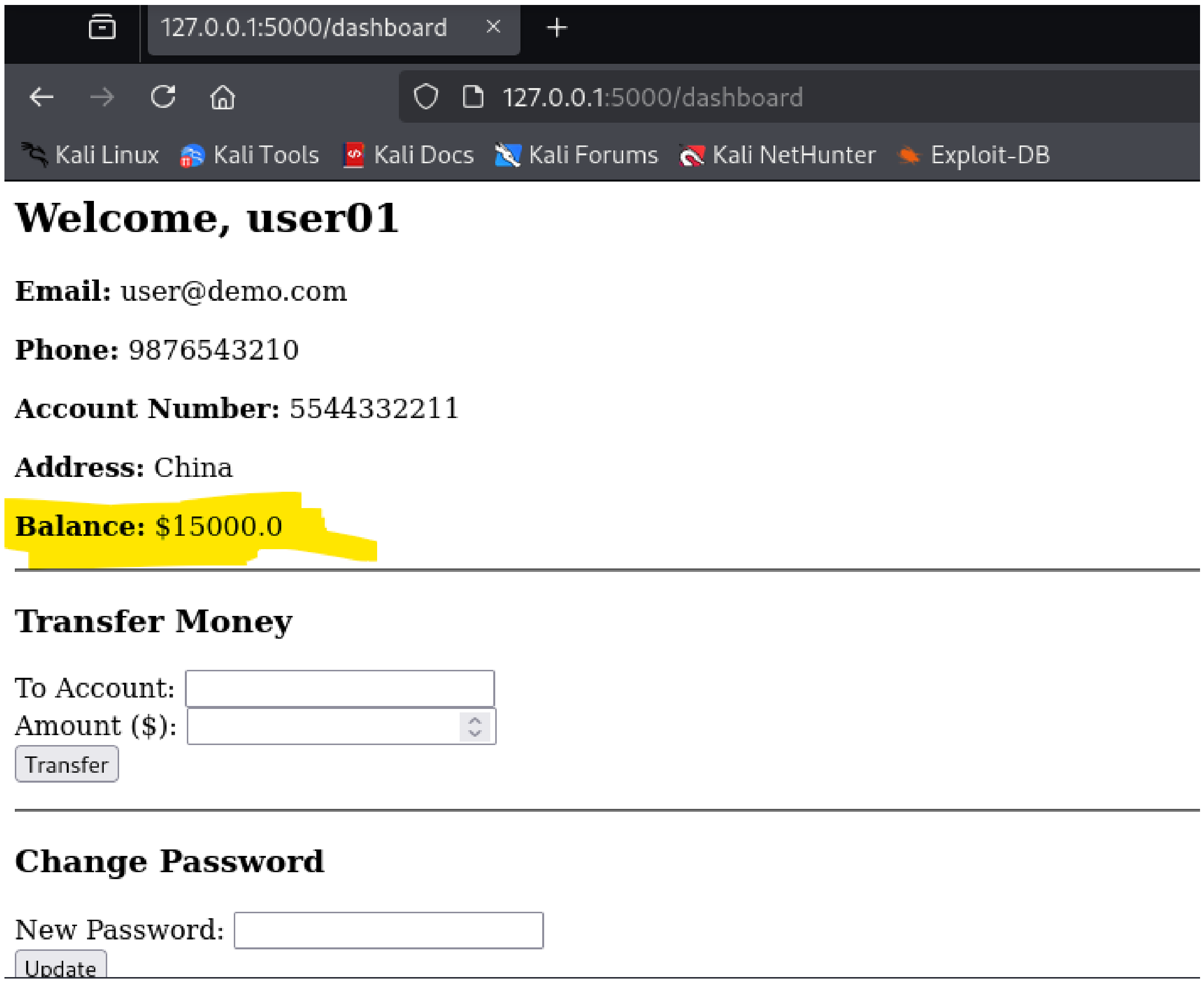
Update

Figure 25:Trying to transfer money to different account.



Transferred $10000.00 to 472384837481 successfully. Back

Figure 26:Successfully transferred.



Welcome, user01

Email: user@demo.com

Phone: 9876543210

Account Number: 5544332211

Address: China

Balance: $15000.0

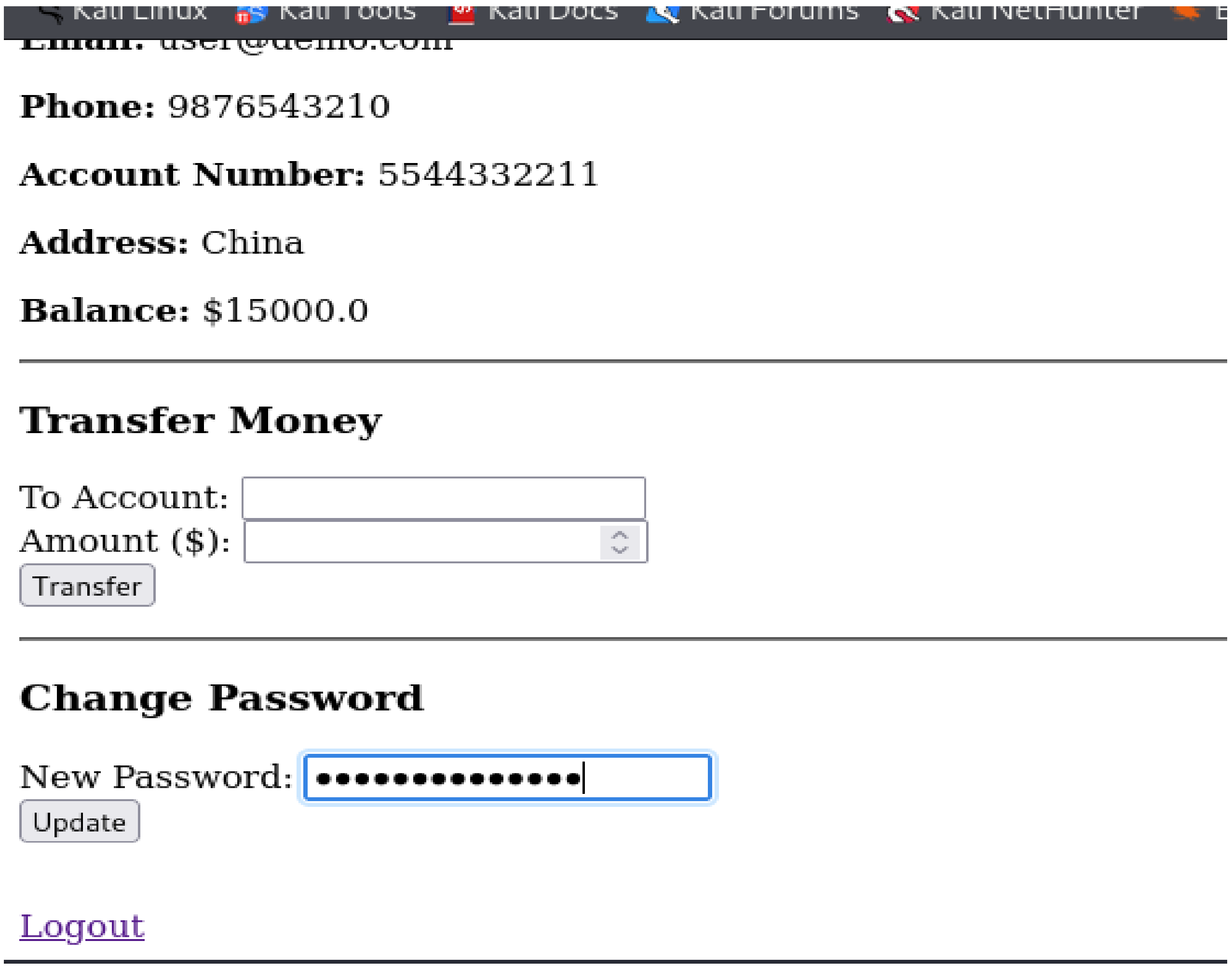
Transfer Money

To Account:
Amount ($):
Transfer

Change Password

New Password:
Update

Figure 27:The amount is now decreased.



Phone: 9876543210

Account Number: 5544332211

Address: China

Balance: $15000.0

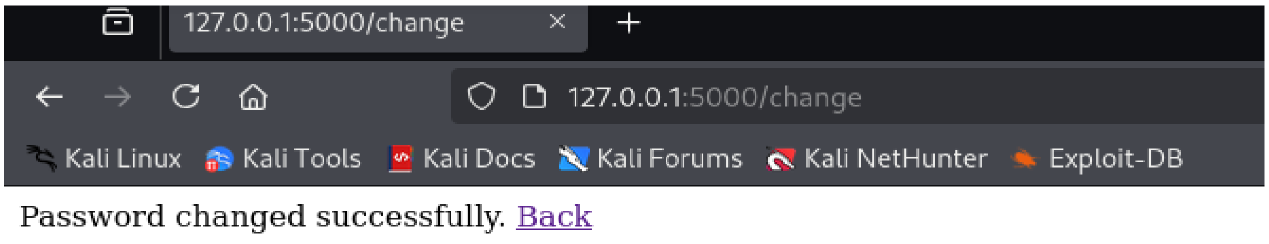
Transfer Money

To Account:
Amount ($):

Change Password

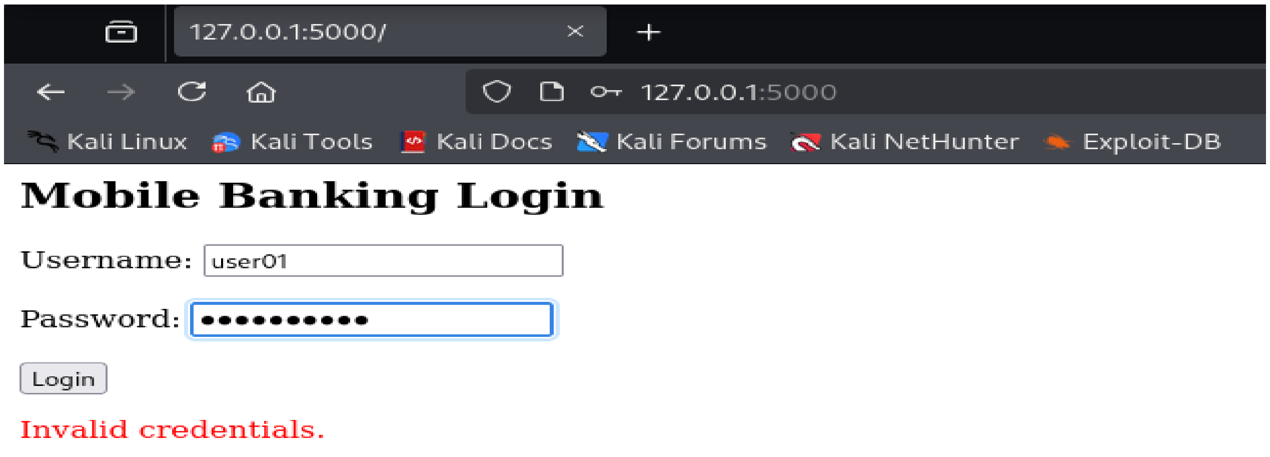
New Password: ●●●●●●●●●●●●●●

Figure 28:Trying to change password.



Password changed successfully. Back

Figure 29:Successfully changed.



Mobile Banking Login

Username: user01

Password: ●●●●●●●●●●●

Invalid credentials.

Figure 30:Trying previous credentials.

3.7 Reporting

On the demonstration, we could successfully crack two pairs of credentials (admin: password123, user: letmein) using a dictionary attack. After successful login, we were able to access the fake inbox system. This outcome highlights that weak passwords and the lack of sound security controls can result in unauthorized access quite effortlessly.

Vulnerability Identified

- The front end of the web application has no account lockout mechanisms, CAPTCHA and multi factor authentication (MFA).
- Since login attempts are unrestricted and password hygiene is poor, login form is vulnerable to dictionary and brute force attack.
- Tools that tested many credentials can bypass detection without limiting or monitoring the rate, due to absence of rate limiting or monitoring.

Impact Assessment

- The system is also opened to possible botnet activity or the deployment of malicious software due to this vulnerability.

- Unauthorized access to user accounts is successful, which shows a very high risk of data breach, identity theft or further lateral movement in the real environment.
- If such weaknesses were in a real system, it could damage the word’s reputation, cause financial loss, and results in bad news under the privacy rules of the country.

4. Mitigation

Since user authentication mechanism is prone to unauthorized access, it is critical to mitigate dictionary attacks. Weak, commonly used passwords, and either poorly implemented security policies or some at tempts at controlling passwords are used as attack targets. Consequently, it is essential to prevent such attacks through a combination of good technical controls and user education.

4.1 Active Mitigation Recommendations

1. Multi Factor Authentication

A Multi Factor Authentication (MFA) is a very powerful security, adding another similar security safety to the username and password. However, even if an attacker is successful in guessing or retrieving a user's password through a dictionary attack, they would still need access to the second verification factor to achieve access to the account.

Examples of commonly used MFA methods include:

- One-Time Passwords (OTP) sent via SMS or email, which expire after a short time.
- Authenticator apps like Google Authenticator, Microsoft Authenticator, which generate secure time-based codes.
- Biometric authentication like fingerprint scans or facial recognition.

MFA reduces the chances of a successful dictionary attack by eliminating the need for only the password alone. As a result, MFA is by far one of the best and most highly recommended methods to counteract unauthorized access to user accounts.

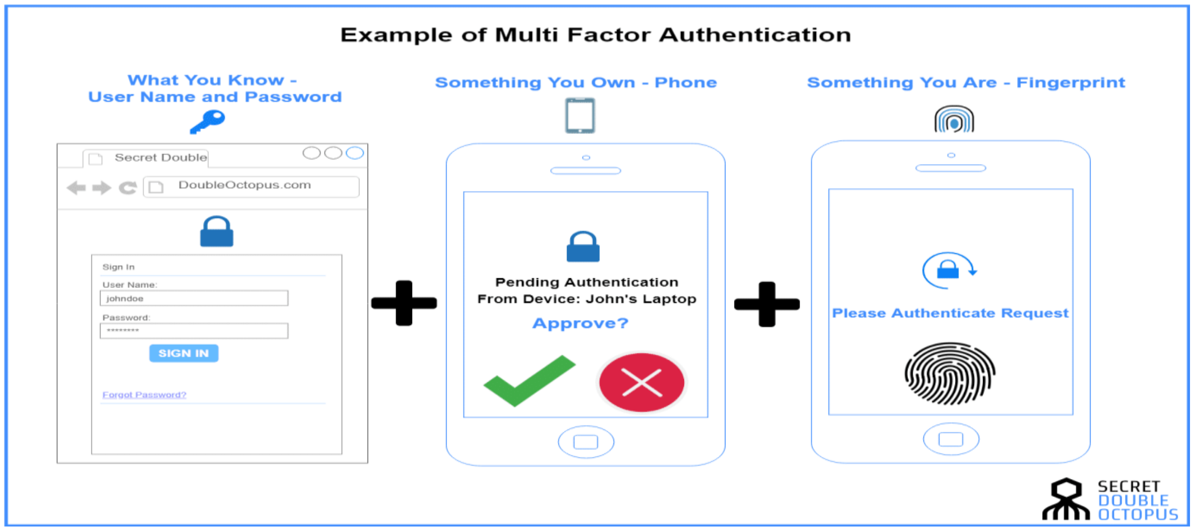

Example of Multi Factor Authentication
What You Know - User Name and Password
Something You Own - Phone
Something You Are - Fingerprint
Secret Double
DoubleOctopus.com
Sign In
User Name:
johndoe
Password:
SIGN IN
Forgot Password?
Pending Authentication From Device: John's Laptop
Approve?
Please Authenticate Request
SECRET DOUBLE OCTOPUS

Figure 31: Example of multi factor authentication (Octopus, 2025).

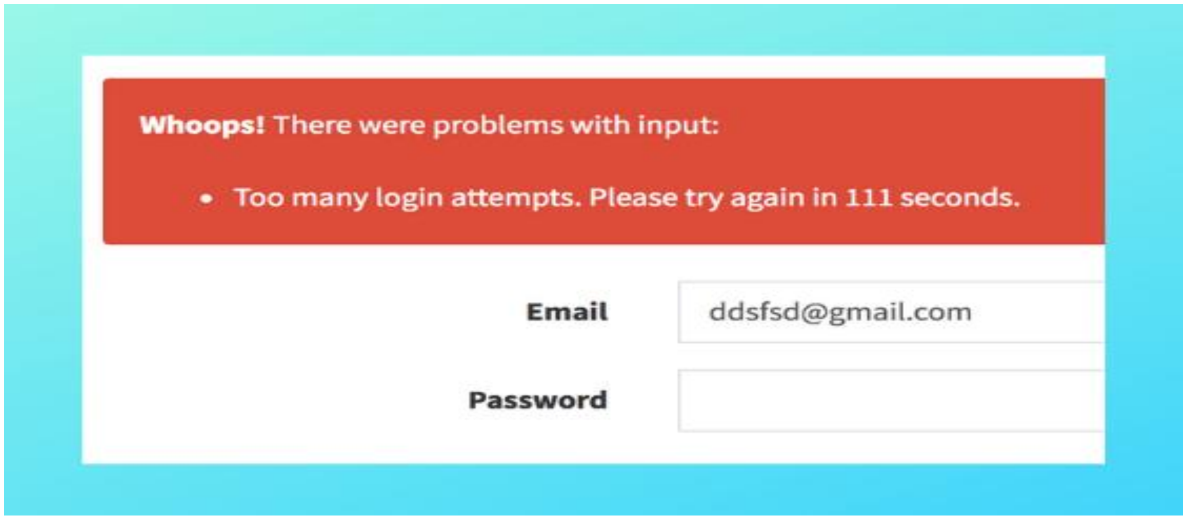
2. Account Lockout and Rate Limiting

Account lockout is a simple yet effective means of protection from dictionary attacks. It temporarily freezes a user's account after a specified number of failed logins, which discourages attacker’s from submitting multiple passwords guesses at high speed.

For instance, locking up an account for 10 seconds after 4 failed attempts making brute-force tools very inconvenient. The mechanism is also used to slow down or even prevent automated attacks against login subsystems.and also tools like hydra, Wfuzz, medusa which use all possible username and password form dictionary are failed to break the login pages.

Rate limiting is a strategy for limiting network traffic. Rate limiting restricts how many login attempts an IP address or user can attempt within a given time frame. Restricting to 5 attempts per one minute.

For example, it allows attackers not to be able to make multiple rapid attempts at many passwords. Libraries like Flask-Limiter can implement this quite simply in web applications. Reducing the number of allowable requests, rate limiting inserts a helpful delay that makes brute-force attacks less effective and less fast.


Whoops! There were problems with input:

- Too many login attempts. Please try again in 111 seconds.

Figure 32: Example of account lockout (Daily, 2022).

3. CAPTCHA on Login Pages

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) is an effective method of preventing automated login attacks by asking users a challenge that can only be solved by a human. Having CAPTCHA on login pages prevents bots and brute-force tools from making multiple requests.

For example, using reCAPTCHA on a Flask login form can stop attackers who use script or tools such as Hydra and Wfuzz. CAPTCHA acts as a gatekeeper that allows only legitimate users to access the authentication system.

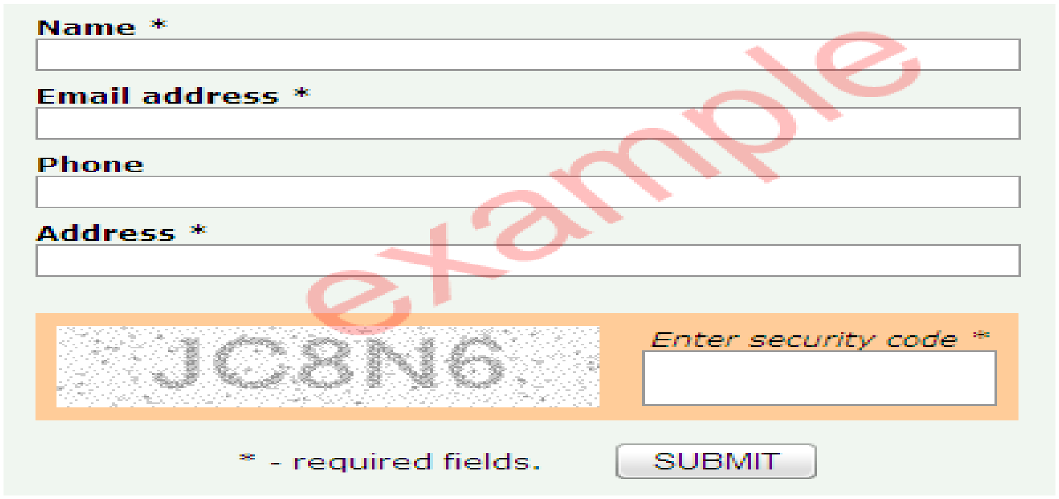


Figure 33: Example of Captcha (snaphost, 2025).

4. Monitoring and Alerting

Real-time monitoring and alerting systems play a significant role in identifying and responding to anomalous login patterns in real-time. Failed attempts at login, unusual IP access patterns, or repeated lockouts can be traced, and admins can be alerted to likely brute force or dictionary attacks.

For instance, pairing log monitoring with alerting techniques like email notification or SIEM systems allows quick response to incipient attacks. Continuous monitoring gives visibility and enables active security response to protect sensitive user information.

4.2 Passive Mitigation Recommendations

1. Enforce Strong Password Policies

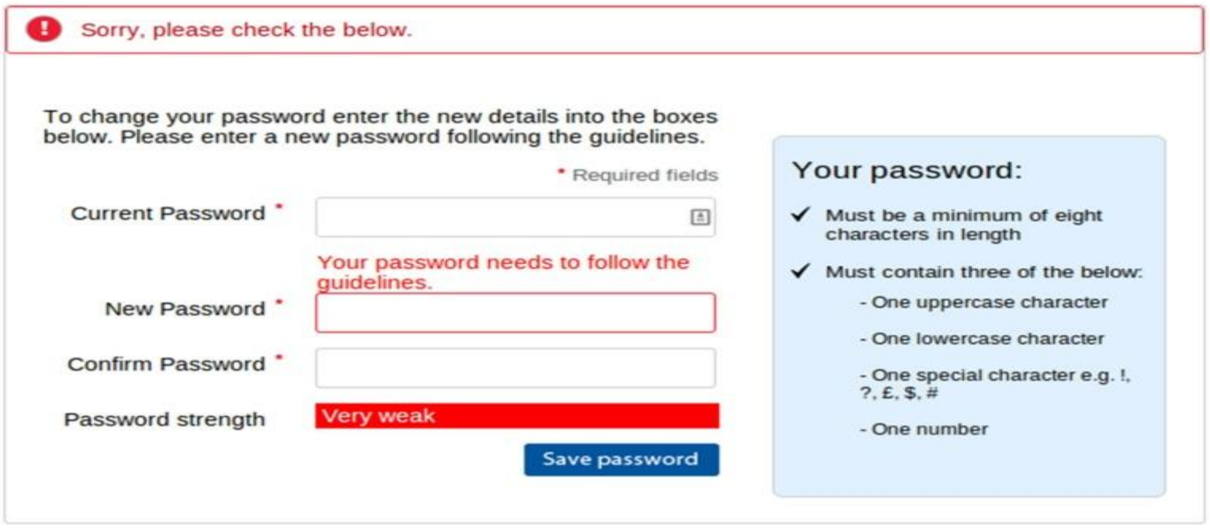
Strong password policies are one of the most effective methods of slowing down the success of dictionary attacks. Passwords should:

Be a minimum of 12–16 characters.

Use of uppercase, lowercase, numbers and special characters.

Beware of using dictionary, names, and any repeated pattern.

Organizations should also disallow a user to use previously compromised passwords by integrating with services like Have I Been Pwned.


Sorry, please check the below.

To change your password enter the new details into the boxes below. Please enter a new password following the guidelines.

* Required fields

Current Password *

Your password needs to follow the guidelines.

New Password *

Confirm Password *

Password strength: Very weak

Your password:

✓ Must be a minimum of eight characters in length

✓ Must contain three of the below:
- One uppercase character
- One lowercase character
- One special character e.g. !, ?, £, $, #
- One number

Figure 34: Strong password policies (Pashley, 2014).

2. Securing Password Storage

Protecting passwords during storage is necessary in the protection of user information, especially if a leak occurs. Storing passwords in plain text should be avoided, and the system can use robust cryptographic hashing algorithms like bcrypt, Argon2, or PBKDF2. These algorithms make reversing or guessing passwords computationally expensive even when the database leaks.

For example, using bcrypt with salting causes even the same passwords to appear different in the database. Secure storage of passwords greatly minimizes the risk of attackers using stolen credentials to break into systems via dictionary or rainbow table attacks.

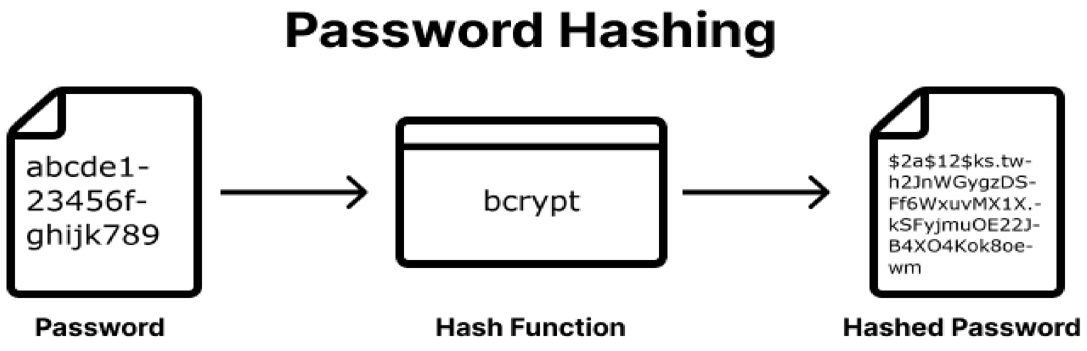


Figure 35: Example of password hashing (authgear, 2025)

5. Evaluation

5.1 Effectiveness of Mitigation Strategies

Applying Multi-Factor Authentication (MFA)

- It provides an extra security layer behind the password.
- Minimize the risk of unauthorized access, even though personal information got leaked.

Strong Password Policies

- Avoid using weak, well-known password which are frequently found in dictionary files.
- Enables proper password security among users.

Account Lockout and Rate Limiting

- Frequently unsuccessful attempts at the same Ip address are effectively bound.
- Helps in identifying and stopping brute-force tools such as Hydra or Medusa.
- Attackers might use specific authorized block users.

CAPTCHA and Behaviour analysis

- It becomes more difficult to block automated login attempts by AI programs.
- Login attempts from unusual access way made unfamiliar time or places are tracked by
- behaviour analysis system.

Monitoring and Alerting

- It allows detection of suspicious activity.
- Helps in quick incident response and forensic analysis

Securing Password Storage

- Secure password storage using bcrypt, Argon2, or PBKDF2 makes it hard to reverse passwords even if stolen which is also known as password hashing or salting.
- It protects user credentials in case of data breach.

5.2 Limitations of Mitigation Strategies

Applying Multi-Factor Authentication (MFA)

- Applying too many security levels may be overwhelmed by users like MFA, CAPTCHA and frequent password change.
- Users may view these measures as obstacles especially when logging.

Strong Password Policies

- Users often write passwords on sticky notes that make the file more unsecured due to frequent change or complex password
- Saving login password in unsafe files or browsers.

Account Lockout and Rate Limiting

- Blocking users that allows for denial-of-service.
- Temporary lockouts may frustrate to user.

CAPTCHA and Behaviour analysis

- It may affect user experience especially visually challenged user.
- Advance bots can bypass basic CAPTCHAs sometimes.

Monitoring and Alerting

- It can bring about false alarms and alert fatigue.
- It needs dedicated resources and adequate changes.

Securing Password Storage

- Proper tools like poor hashing still creates threats.
- It increases difficulty in development and maintenance.

5.3 Overall Mitigation Strategies

There are various methods for mitigating cybersecurity, including Multi-Factor Authentication (MFA), strong password policies, account lockout mechanism, CAPTCHA with behavior analysis, monitoring and alerting, as well as secure storage of passwords. These methods ensure unauthorized access, identify suspicious activities and user credentials. However, they may also induce usability issues, false alerts or implementation complexity. Despite these constrictions, they are used everywhere, from banking, to corporate systems, to web platforms to protect sensitive data and systems.

5.4 Application areas of Mitigation Techniques

Applying Multi-Factor Authentication (MFA)

- It is used in online banking system to protect transactions and prevent from unauthorized
- fund transfer.
- It is applied in email services e.g. Gmail, Outlook to prevent against account breach.
- It is used in healthcare to ensure protection of sensitive medical records of patients abides to HIPPA standards.

Strong Password Policies

- It is commonly used in business networks and services that manage sensitive data.
- It is used in educational system as a way of securing student records and academic research data.
- It is used in database management systems and for administrative of serve for privileged accounts.

Account Lockout and Rate Limiting

- It is applied in login system of web applications and admin panels.
- It is used in mobile banking apps to limit pin/password guessing attempts.
- It is used in industrial control system to prevent unauthorized access to operational technology.

CAPTCHA and Behaviour analysis

- It is used in comment sections to filter out spam message automatically.
- It is used in website login forms to block automated bot attacks.
- It is used banking sector to verify real human transactions.

Monitoring and Alerting

- It is used in security Operation Centres and industrial IT environments.
- It is used in industrial IT environments to protect infrastructure form cyber-attack.

- It is used in smart home devices to prevent unauthorized control of IoT systems.

Securing Password Storage

- It is used for authentication in websites and mobile applications.
- It is used in government databases to protect sensitive citizens information.
- It is use in cloud storage services to ensure files remain private and secure.

6. Conclusion

In today’s digital world, dictionary attacks is some of the most persistent and dangerous threats to individuals, businesses, and even government systems. This report explored the mechanisms behind these attacks, particularly focusing on dictionary attacks which guess predictable passwords to gain unauthorized access. From this report, we demonstrated how these attacks operate using real tools like Flask, Nmap, and Wfuzz in a controlled lab environment. This test not only simulated real-world scenarios but also highlighted key vulnerabilities, including the absence of security controls such as CAPTCHA, account lockout, Multi-Factor Authentication (MFA) and others.

Our findings highlight the need for a multi-layered security defence against dictionary attacks. While these threats remain common effective countermeasures like Multi -factor authentication (MFA), strong password policies, CAPTCHA, rate limiting, and secure password storage using modern hashing algorithms like bcrypt these measures, when combined with continuous monitoring and alerting systems, can significantly reduce the risk posed by brute force techniques. The evaluation section also highlighted that while these strategies are effective, they may introduce usability challenges and require ongoing maintenance. Nevertheless, their widespread application across industries from banking to healthcare and cloud services emphasizes their importance in safeguarding digital environments.

In conclusion, cybersecurity is not only a technological challenge but a collective responsibility that includes proactive policy implementation, technical safeguards, and end-user awareness. As attackers keep getting smarter, using AI and other sophisticated tools to enhance brute force techniques faster than ever before, Organizations must also adapt by reinforcing their systems through ethical penetration testing, routine audits, and updated security frameworks like PTES. Ultimately, the insights from this study highlight the urgency of staying careful and continuously improving cybersecurity measures to protect sensitive data, maintain trust, and ensure digital resilience in an ever-changing threat landscape. The path forward requires new perspectives on approaching cybersecurity from that of a cost to strategic investment in the long-term survival if the organization and reputation.

7. Reference

[1]. Alaa, N. & AL-Shareefi, F., 2024. A Comparative Study Between Two Cybersecurity Attacks: Brute Force and Dictionary Attacks. *Journal of Kufa for Mathematics and Computer,* Volume 11, pp. 133--139.

[2]. Aslam, F. A., Mohammed, H. N. & Lokhande, P. S., 2015. Efficient Way Of Web Development Using Python And Flask.. *International Journal of Advanced Research in Computer Science,* Volume 6.

[3]. Aslan, O. et al., 2023. A comprehensive review of cyber security vulnerabilities, threats, attacks, and solutions. *Electronics,* Volume 12, p. 1333.

[4]. authgear, 2025. *Example of password hashing.* (authgear).

[5]. Ayankoya, Folasade, Ohwo & Blaise, 2019. Brute-force attack prevention in cloud computing using one-time password and cryptographic hash function. *International Journal of Computer Science and Information Security (IJCSIS),* Volume 17, pp. 7--19.

[6]. Ba, M. H. N., Bennett, J., Gallagher, M. & Bhunia, S., 2021. A case study of credential stuffing attack: Canva data breach. In: *2021 International Conference on Computational Science and Computational Intelligence (CSCI).* s.l.:s.n., pp. 735--740.

[7]. B., L. a. S. & J. a. B., 2018. Brute-force and dictionary attack on hashed real-world passwords. In: *2018 41st international convention on information and communication technology, electronics and microelectronics (mipro).* s.l.:s.n., pp. 1161--1166.

[8]. Bosnjak, L., Sres, J. & Brumen, L., 2018. Brute-force and dictionary attack on hashed real-world passwords. In: *2018 41st international convention on information and communication technology, electronics and microelectronics (mipro).* s.l.:s.n., pp. 1161--1166.

[9]. Castagnaro, A., Conti, . M. & Pajola, . L., 2024. Offensive AI: Enhancing Directory Brute-forcing Attack with the Use of Language Model. In: *Proceedings of the 2024 Workshop on Artificial Intelligence and Security.* s.l.:s.n., pp. 184--195.

[10]. Crane, C., 2021. *Brute Force attack.* (The SSL Store).

[11]. Curtin & Matt, 2005. Brute Force. In: *Brute force.* s.l.:Springer.

[12]. Daily, L., 2022. *Example of account lockout.* (Laravel Daily).

[13]. Duffy, C., 2015. Learning penetration testing with Python. In: *Learning penetration testing with Python.* s.l.:Packt Publishing Ltd.

[14]. Lashkari, A. H., Ghaffari, N. & Ghorbani, A. A., 2014. A Survey on Password Attacks: Classic and Modern Approaches to Password Security. *International Journal of Network Security & Its Applications (IJNSA),* 6(3), p. 21–39.

[15]. Lazauskas, R., 2018. Nano Server and Containers in Windows Server 2016.

[16]. Marvin, R., 2013. *sdtimes.* [Online]
Available at: https://sdtimes.com/adobe/sd-times-blog-passwords-stolen-in-adobe-breach-were-encrypted-not-hashed/
[Accessed 19 04 2025].

[17]. Octopus, S. D., 2025. *Example of multi factor authentication.* (Secret Double Octopus).

[18]. Osuagwa, N., 2024. *techbehindfintech.* [Online]
Available at: https://techbehindfintech.com/data-breaches/how-hybrid-password-attacks-work-and-how-to-defend-against-them
[Accessed 19 04 2025].

[19]. Owens, Jim, Matthews & Jeanna, 2008. A study of passwords and methods used in brute-force SSH attacks. In: *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET).* s.l.:s.n., p. 8.

[20]. Pashley, D., 2014. *Strong password policies.* (David Pashley).

[21]. Pinkas, B. & Sander, T., 2002. Securing passwords against dictionary attacks. In: *Proceedings of the 9th ACM Conference on Computer and Communications Security.* s.l.:s.n., pp. 161--170.

[22]. Ramesh, N. C. & Lan, Y.-P., 2024. LinkedIn Data Hack. In: *Information Technology Security and Risk Management.* s.l.:CRC Press, pp. 31--38.

[23]. S. et al., 2023. Security vulnerability analysis using penetration testing execution standard (PTES): case study of government's website. In: *Proceedings of the 2023 6th international conference on electronics, communications and control engineering.* s.l.:s.n., pp. 139--145.

[24]. Shah, M. et al., 2019. Penetration testing active reconnaissance phase--optimized port scanning with nmap tool. In: *2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET).* s.l.:s.n., pp. 1--6.

[25]. snaphost, 2025. *Example of Captcha.* (snaphost).

[26]. Unstop, 2024. *What is Dictionary Attack in Cybersecurity Explained.* (Unstop.com).

[27]. van Rooij, O. et al., 2021. webfuzz: Grey-box fuzzing for web applications. In: *Computer Security--ESORICS 20226th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4--8, 2021, Proceedings, Part I 26.* s.l.:s.n., pp. 152--172.

[28]. Wang, X., Yan, Z., Zhang, R. & Zhang, P., 2021. Attacks and defenses in user authentication systems: A survey. *Journal of Network and Computer Applications,* Volume 188, p. 103080.

[29]. Wickramasinghe, S., 2024. *Brute Force Attacks: Techniques, Types & Prevention.* (Splunk).

[30]. Zhang, L., Tan, C. & Yu, F., 2017. An improved rainbow table attack for long passwords. *Procedia Computer Science,* Volume 107, pp. 47--52.

8. Appendix

8.1 Code of website

Our fake mobile bank login was created by tool called flask using python and html both.

Here,

```
from flask import Flask, render_template_string, request, redirect, session



app = Flask(__name__)

app.secret_key = 'demo123'



# Two hardcoded user accounts

users = {

    "admin": {

        "password": "password123",

        "email": "admin@demo.com",

        "phone": "9810174579",

        "account": "0011223344",

        "address": "Newyork",

        "balance": 50000

    },

    "user01": {

        "password": "letmein123",

        "email": "user@demo.com",

        "phone": "9876543210",
        "account": "5544332211",

        "address": "China",

        "balance": 25000

    }

}


# HTML template for login page

login_page = '''

<h2> Mobile Banking Login</h2>

<form method="post">

  Username: <input type="text" name="username"><br><br>

  Password: <input type="password" name="password"><br><br>

  <input type="submit" value="Login">

</form>

{% if error %}<p style="color:red;">{{ error }}</p>{% endif %}

'''


# HTML template for dashboard

dashboard_page = '''

<h2>Welcome, {{ user }}</h2>

<p><b>Email:</b> {{ data.email }}</p>

<p><b>Phone:</b> {{ data.phone }}</p>

<p><b>Account Number:</b> {{ data.account }}</p>
```

```
<p><b>Address:</b> {{ data.address }}</p>

<p><b>Balance:</b> ${{ data.balance }}</p>


<hr>

<h3> Transfer Money</h3>

<form method="post" action="/transfer">

  To Account: <input type="text" name="to"><br>

  Amount ($): <input type="number" name="amount"><br>

  <input type="submit" value="Transfer">

</form>


<hr>

<h3> Change Password</h3>

<form method="post" action="/change">

  New Password: <input type="password" name="newpass"><br>

  <input type="submit" value="Update">

</form>


<br><a href="/logout"> Logout</a>

'''


@app.route('/', methods=['GET', 'POST'])

def login():
    error = ''
    if request.method == 'POST':
        uname = request.form['username']
        pword = request.form['password']
        if uname in users and users[uname]['password'] == pword:
            session['user'] = uname
            return redirect('/dashboard')
        else:
            error = 'Invalid credentials.'
    return render_template_string(login_page, error=error)


@app.route('/dashboard')
def dashboard():
    if 'user' not in session:
        return redirect('/')
    uname = session['user']
    return render_template_string(dashboard_page, user=uname, data=users[uname])


@app.route('/transfer', methods=['POST'])
def transfer():
    if 'user' not in session:
        return redirect('/')
    uname = session['user']
    amount = float(request.form['amount'])
    if amount > users[uname]['balance']:
        return "<p style='color:red;'> Insufficient balance. <a href='/dashboard'>Back</a></p>"
    users[uname]['balance'] -= amount
    return f"<p> Transferred ${amount:.2f} to {request.form['to']} successfully. <a
href='/dashboard'>Back</a></p>"

@app.route('/change', methods=['POST'])
def change_password():
    if 'user' not in session:
        return redirect('/')
    uname = session['user']
    users[uname]['password'] = request.form['newpass']
    return "<p> Password changed successfully. <a href='/dashboard'>Back</a></p>"

@app.route('/logout')
def logout():
    session.clear()
    return redirect('/')

if __name__ == '__main__':
   app.run(debug=True)
```

8.2 Code for hosting website

```
python3 fake_login.py
```

8.3 Code to perform bruteforce attack

Command used for attack ,

```
 wfuzz -z file,username.txt -z file,password.txt\

-d "username=FUZZ&password=FUZZ2" \

--hc=403 http://127.0.0.1:5000/login
```

8.3 code for hosting website