Project Report: SSH Cracker

1. Objective

The objective of this project was to simulate a **brute-force attack on SSH services** using custom Python code. This tool was developed to ethically demonstrate how weak authentication systems can be compromised through dictionary attacks. It allows a red team analyst to automate login attempts using a known username and a list of possible passwords. The primary focus of the tool is to help identify and exploit poor credential management, and to understand the effectiveness of brute-force prevention mechanisms in a controlled environment.

2. Project Overview

The **SSH Brute-Force Cracker** is a multi-threaded Python-based tool that attempts to log in to a target SSH server by iterating over a user-defined password list. The tool uses the `paramiko` library to establish SSH sessions and `threading` to conduct multiple login attempts in parallel, improving speed and efficiency.

It accepts the target IP address, username, password wordlist, and an optional number of threads as command-line arguments. If a valid password is found, the tool stops execution and reports the working credentials.

This project serves both as an offensive security utility and an educational demonstration of how critical strong password policies and SSH hardening are to securing remote systems.

3. Key Features

- **Automated SSH login attempts** using a user-supplied dictionary file.
- **Multi-threading** support for parallel processing of password guesses, significantly reducing time required to brute-force.
- **Robust exception handling** for failed connections, timeouts, or blocked attempts.
- **Credential logging** upon successful login detection.
- **Port customization**, allowing testing against SSH services running on non-standard ports.

4. Tools & Technologies Used

Component	Purpose
Python 3.x	Primary scripting language
Paramiko	SSH protocol implementation for establishing sessions
Threading	Enables concurrent login attempts for faster processing
Argparse	Provides command-line argument handling for flexible tool usage

5. Architecture & Workflow

The workflow of the SSH Cracker tool proceeds in the following steps:

1. **User Input**: The user provides the target IP address, a known username, and a password wordlist via the command line.
2. **Connection Attempts**: The script reads passwords from the list and uses Paramiko to try logging into the SSH server using each one.
3. **Thread Pool Execution**: A pool of threads is created to attempt logins concurrently.
4. **Result Handling**: If a successful login occurs, the password is printed and saved. Execution is stopped.
5. **Exception Handling**: Incorrect credentials, connection timeouts, or authentication errors are gracefully managed and logged.

The tool is designed for use in a **closed lab environment** against **authorized SSH services** only.

6. Ethical Considerations

This project was developed and tested entirely within a **safe and isolated virtual lab environment**. The SSH targets were virtual machines pre-configured with known user accounts and dummy passwords. No external systems or production servers were involved at any stage of development or testing.

This tool is intended strictly for:

- Penetration testing simulations
- Credential policy audits
- Ethical hacking education
- Red team exercise development

Use of this tool against systems without explicit permission is a violation of ethical hacking principles and could be illegal.

7. Learning Outcomes

Through the development and testing of this project, I gained hands-on experience in:

- Designing and building **authentication attack tools** from scratch
- Working with the **SSH protocol** and secure session handling in Python
- Understanding the risks of **weak passwords** and the ease of exploitation
- Implementing **threading for concurrent execution**, enhancing tool performance
- Managing **socket errors, authentication failures, and race conditions** gracefully

Additionally, this project helped solidify my understanding of **real-world attacker techniques** and defensive strategies to mitigate them, such as rate limiting, account lockouts, and anomaly detection.

8. Conclusion

The SSH Brute-Force Cracker project effectively demonstrated the risk of using weak or guessable passwords in remote login systems. By simulating a common attack vector, the tool showcases how quickly an attacker can compromise systems with poor credential hygiene. The hands-on development process provided critical insight into **ethical hacking methodologies**, the limitations of brute-force attacks, and the importance of enforcing **strong password policies** and **SSH access controls**.

This project contributes to my foundational understanding of offensive security practices and improves my ability to build tools for red-team simulations and security assessments.