Project Report: FTP Cracker

1. Objective

The objective of this project was to develop a tool that simulates a **brute-force attack against FTP services** using Python. The project aimed to demonstrate how insecure password practices can be exploited over plaintext protocols such as FTP. By implementing automated login attempts using a username and a password dictionary, this tool serves as a practical learning experience in offensive security, credential auditing, and protocol-level attack simulation.

2. Project Overview

The **FTP Cracker** is a command-line Python tool that automates the process of attempting login credentials against an FTP server using the standard `ftplib` library. The tool reads a list of passwords from a file and attempts to authenticate using each one with a given username.

The goal is to identify weak or default credentials that would allow unauthorized access to FTP servers. This simulation can help reinforce the importance of strong authentication policies and proper service configuration.

The script includes robust exception handling, response checking, and output logging to ensure effective and reliable testing in a lab environment.

3. Key Features

- **Automated login attempts** against FTP services using a user-provided username and password wordlist.
- **Customizable port support**, allowing targeting of FTP servers running on non-default ports.
- **Efficient password list parsing**, skipping failed attempts and stopping on the first successful match.
- **Error handling and retry logic** for common FTP errors (timeouts, login failures, broken pipes).
- **Command-line interface** for easy and flexible usage.

4. Tools & Technologies Used

Component	Purpose
Python 3.x	Main scripting language
ftplib	Python's built-in FTP library used for connection and login
argparse	Command-line parsing for flexible tool usage
os/time	For file handling and optional delay between attempts

5. Workflow

The cracking process proceeds as follows:

1. **Input Parameters**: The user provides the target IP address, username, password wordlist, and optionally the port via command-line arguments.
2. **Connection Handling**: The script uses `ftplib.FTP()` to connect to the FTP service.
3. **Brute-Force Loop**: For each password in the file:
    - The tool attempts a login using the provided username and current password.
    - On success, the valid credentials are printed and saved.
    - On failure, the next password is tested.
4. **Success or Exhaustion**: The process ends when either a valid password is found or the list is fully tested.

6. Architecture Overview

- The user runs `ftp_cracker.py` with the target IP, username, and password list.
- The script reads passwords from the file line by line.
- For each password:
    - Attempts to connect and log in to the FTP server using `ftplib`.
    - On success: prints valid credentials and stops.
    - On failure: moves to the next password.
- Handles common FTP errors like timeouts and login failures.
- Runs entirely against a **controlled lab FTP server** for ethical testing.

7. Ethical Considerations

This project was designed and tested exclusively in a **controlled environment** using a virtual FTP server with known credentials. No external or unauthorized systems were targeted.

The project serves the following ethical and educational purposes:

- Teaching brute-force detection and prevention
- Understanding protocol weaknesses in FTP
- Training in safe offensive security tool development

This tool must **never be used on live or unauthorized systems**.

8. Learning Outcomes

By developing this FTP brute-force tool, I gained:

- A clear understanding of how **FTP authentication** can be tested programmatically.
- Practical knowledge of **password enumeration** and **how default credentials are exploited**.
- Experience handling network-related exceptions and server timeouts.
- Familiarity with the **limitations of plaintext protocols** like FTP.
- Skills in writing **robust, flexible command-line tools** using Python.

This project also improved my approach to secure tool design, emphasizing responsible coding practices and ethical limitations.

9. Conclusion

The FTP Cracker project was a valuable exercise in demonstrating one of the oldest but still relevant attack techniques: brute-force credential guessing over insecure services. Through this project, I gained insights into how red teamers exploit weak login systems and how defenders can build stronger countermeasures, such as enforcing account lockouts, disabling anonymous logins, and avoiding FTP in favor of secure alternatives like SFTP.

This project served as a stepping stone to understanding **network protocol exploitation**, **authentication hardening**, and **offensive tool development** in cybersecurity.