Prerequisites for the FTP Cracker Project

By: **Inlighn Tech**

Overview

The FTP Cracker project requires students to develop a Python script that brute-forces FTP server credentials by testing username-password combinations. Two solutions are provided: ftp_brute.py (basic, with a hardcoded host, user, and password list, using multithreading) and advanced_ftp_brute.py (advanced, with flexible arguments, password generation, and userlist support). Both use the ftplib library for FTP connections, threading and queue for parallel execution, and colorama for colored output, with the advanced version adding argparse, itertools, and string. To write these scripts independently, students must master several key concepts and skills from the curriculum. These prerequisites, paired with practical Python code examples, ensure students understand how FTP cracking works and can implement either version effectively.

Prerequisite Knowledge and Skills

1. **FTP Fundamentals – File Transfer Protocol Overview**

- **What to Know**:
  - FTP (File Transfer Protocol) enables file transfers over a network, typically on port 21, using credentials (username/password).
  - Brute-forcing tests combinations to find valid credentials, exploiting weak security.
- **How It Applies**:
  - Both scripts use ftplib.FTP() to attempt FTP logins, checking if credentials work (connect_ftp()).
- **Curriculum Source**:
  - Not explicitly listed; assumes basic networking knowledge from "NETWORKING, SOCKETS, AND CYBERSECURITY" PDFs (e.g., "Understanding Ports, IP Addresses, and Protocols in Cybersecurity.pdf").
- **Practical Prep**:
  - Test FTP manually with ftp 127.0.0.1 (requires a local FTP server like vsftpd).
- **Python Code Block**:

```
# Basic FTP connection test
import ftplib

host = "127.0.0.1"  # Localhost
user = "testuser"   # Replace with a valid user
password = "wrong"  # Replace with a test password
try:
    with ftplib.FTP() as server:
        server.connect(host, 21, timeout=5)
        server.login(user, password)
        print(f"Connected to {host} with {user}:{password}")
except ftplib.error_perm:
    print(f"Failed login for {user}:{password}")
```

-
  - **Run It**: Save as ftp_test.py, run with python ftp_test.py (requires a local FTP server). Introduces FTP login attempts, like connect_ftp().

2. **File Input/Output (I/O) Basics**

- **What to Know**:
  - Read text files (e.g., wordlists) with open() in text mode ('r'); process lines with splitlines() or split().
- **How It Applies**:
  - Both scripts read password lists (wordlist.txt or load_lines()); the advanced version also reads usernames.
- **Curriculum Source**:
  - "PYTHON BASICS" Lesson 11 (IO with Basic Files).
- **Practical Prep**:
  - Practice file reading.
- **Python Code Block**:

```
# Read a wordlist file
with open("test_pass.txt", "w") as f:
    f.write("pass1\npass2\npass3\n")  # Create sample wordlist

with open("test_pass.txt", "r") as f:
    passwords = f.read().splitlines()
    print("Passwords:", passwords)
```

-
  - **Run It**: Save as file_io_test.py, run with python file_io_test.py. Prepares for password list handling in both scripts.

3. **Exception Handling**
   - **What to Know**:
     - Use try/except to catch FTP errors (e.g., ftplib.error_perm for authentication failures) and general exceptions.
     - Gracefully ignore failed attempts without stopping execution.
   - **How It Applies**:
     - connect_ftp() handles authentication errors (error_perm) and other exceptions to keep brute-forcing robust.
   - **Curriculum Source**:
     - "PYTHON BASICS" Lesson 21 (Errors and Exception Handling).
   - **Practical Prep**:
     - Test FTP error handling.

- **Python Code Block**:

```
# Handle FTP login errors
import ftplib

host = "127.0.0.1"
user = "testuser"
password = "wrong"
try:
    with ftplib.FTP() as server:
        server.connect(host, 21, timeout=1)
        server.login(user, password)
        print("Success")
except ftplib.error_perm:
    print("Authentication failed")
except Exception as e:
    print(f"Error: {e}")
```

-
    - **Run It**: Save as ftp_error_test.py, run with python ftp_error_test.py (requires an FTP server). Mimics connect_ftp() error handling.

4. **Multithreading with threading and Queue**
    - **What to Know**:
        - threading.Thread runs tasks concurrently; queue.Queue manages tasks across threads.
        - Use daemon=True for background threads, q.put()/get() for task distribution, and q.join() to wait for completion.
        - Manipulate queue state (e.g., clear(), all_tasks_done) to stop on success.
    - **How It Applies**:
        - Both scripts use threads and a queue to test credentials in parallel, stopping when credentials are found.
    - **Curriculum Source**:
        - "Network Scanner" Lesson 4 ("Working with Threads for our program.mp4").
    - **Practical Prep**:
        - Test threading with a queue.

- **Python Code Block**:

```
# Use threads and a queue
import threading
import queue
import time

q = queue.Queue()

def worker():
    while not q.empty():
        task = q.get()
        print(f"Processing: {task}")
        time.sleep(0.5)
        q.task_done()

for i in range(3):
    q.put(f"Task {i}")

threads = [threading.Thread(target=worker, daemon=True) for _ in
range(2)]
for t in threads:
    t.start()
q.join()
print("All tasks done")
```

    - **Run It**: Save as thread_queue_test.py, run with python thread_queue_test.py. Prepares for connect_ftp() threading.

5. **Command-Line Arguments with argparse (Advanced Version)**
    - **What to Know**:
        - argparse parses inputs with required (e.g., --host) and optional arguments (e.g., --port, --threads).
        - Use action="store_true" for flags and set defaults.
    - **How It Applies**:
        - The advanced script uses argparse for flexible host, port, user, and password inputs, plus generation options.
    - **Curriculum Source**:
        - "SSH Cracker" Lesson 4 ("User Input.mp4").

- **Practical Prep**:
  - Practice argument parsing.
- **Python Code Block**:

```
# Parse command-line arguments
import argparse

parser = argparse.ArgumentParser(description="Test parser")
parser.add_argument("--host", required=True, help="Target host")
parser.add_argument("--port", type=int, default=21, help="Port")
parser.add_argument("-t", "--threads", type=int, default=2,
help="Threads")
args = parser.parse_args()

print(f"Host: {args.host}, Port: {args.port}, Threads: {args.threads}")
```

  - **Run It**: Save as args_test.py, run with python args_test.py --host 127.0.0.1 -t 3. Prepares for main() in the advanced script.

6. **Password Generation with itertools (Advanced Version)**
   - **What to Know**:
     - itertools.product() generates all combinations of characters; yield creates a generator for memory efficiency.
     - Use string module for character sets (e.g., string.ascii_letters).
   - **How It Applies**:
     - generate_passwords() in the advanced script generates passwords on the fly for brute-forcing.
   - **Curriculum Source**:
     - Implied in "Password Cracker" project; builds on "PYTHON BASICS" function lessons.
   - **Practical Prep**:
     - Test password generation.
   - **Python Code Block**:

```
# Generate passwords with itertools
```

```
import itertools
import string

def gen_passwords(chars, length):
    for combo in itertools.product(chars, repeat=length):
        yield ''.join(combo)

chars = "ab"
for pwd in gen_passwords(chars, 2):
    print(pwd)  # Outputs: aa, ab, ba, bb
```

- **Run It**: Save as generator_test.py, run with python generator_test.py. Prepares for generate_passwords().

7. **Colored Output with colorama**
   - **What to Know**:
     - colorama provides ANSI color codes (e.g., Fore.GREEN) for terminal output; init() ensures cross-platform compatibility.
   - **How It Applies**:
     - Both scripts use colorama to highlight attempts and successful credentials.
   - **Curriculum Source**:
     - Not explicitly taught; assumes a brief intro (common in Python projects).
     - Note: Students need pip install colorama.
   - **Practical Prep**:
     - Test colored output.
   - **Python Code Block**:

```
# Use colorama for colored output
from colorama import init, Fore

init()
print(f"{Fore.GREEN}Success!{Fore.RESET}")
print(f"{Fore.RED}Trying something{Fore.RESET}")
```

        - **Run It**: Install colorama (pip install colorama), save as color_test.py, run with python color_test.py. Prepares for colored feedback in connect_ftp().

How FTP Cracking Works

- **Concept**:
    - FTP cracking attempts to log into an FTP server by testing username-password pairs until a valid combination is found.
    - This simulates attacks on poorly secured FTP servers.
- **Basic Workflow (ftp_brute.py)**:
    - Hardcode host, user, port, and thread count; read passwords from wordlist.txt.
    - Use threads and a queue to test passwords in parallel.
    - Handle authentication errors and stop on success, printing credentials.
- **Advanced Workflow (advanced_ftp_brute.py)**:
    - Accept flexible arguments for host, port, users, passwords, and generation options via argparse.
    - Load or generate credentials, queue them, and test in parallel with threads.
    - Provide detailed feedback and stop on success.
- **Why Multithreading?**:
    - Speeds up brute-forcing by testing multiple combinations simultaneously.

How to Write the FTP Cracker Script

Using these prerequisites, students can build either version:

- **Basic Version (ftp_brute.py)**:
    1. Import ftplib, threading, queue, and colorama.
    2. Define connect_ftp() to test passwords with FTP.login(), handling errors.

3. Hardcode host, user, and port; read passwords and queue them.
4. Launch threads and use q.join() to wait for completion.

- **Advanced Version (advanced_ftp_brute.py)**:
    1. Add argparse, itertools, and string; define load_lines() and generate_passwords().
    2. Enhance connect_ftp() with host/port args and better error handling.
    3. Parse args for host, port, users, passwords, and threads; queue credentials dynamically.
    4. Launch threads and manage with q.join().

Notes for Students

- **Setup**: Install colorama (pip install colorama); no other external libraries needed beyond Python's standard library. Set up a local FTP server (e.g., vsftpd on Linux) and create a small wordlist.txt (e.g., pass1, 123) and userlist.txt (e.g., user1, user2).
- **Engagement**: Run each code block to practice FTP connections, threading, and generation. Test the basic script with python ftp_brute.py (edit hardcoded values) or the advanced one with python advanced_ftp_brute.py --host 127.0.0.1 -u testuser -w wordlist.txt -t 5.
- **Tips**: Use a local server to avoid legal/ethical issues; limit threads to avoid overwhelming the target. Ensure the FTP server is running (sudo systemctl start vsftpd on Linux).

Jump to the FTP Cracker Project Description if you are ready now.