

|  |
| --- |
| **Python Scripting on Enumeration Libraries**  *A tutorial and walkthrough on the construction of a simple and rudimentary python script involving nmap, nikto, and more.*  **Bryce Place**  CMP320: Advanced Ethical Hacking  2024/25 |

*Note that Information contained in this document is for educational purposes.*

Abstract

Downloading several imported packages and software’s, learning each of their specifics, inputting their commands and processes only to grab the same information from several attempts beforehand. The need to streamline a mind numbingly simple process of the penetration testing format is urgent. Having to start up and produce several nmap scan reports, banner grabbing the websites information afterwards, scanning for extensions with dirb, and finally conducting a full Nessus examination. This process could take upwards of 3 hours just to complete. Having a script run all these processes one after the other with minimal user input allots more time to the end user.

Streamlining is a process that takes places in all forms and professions. Turning the workflow into a factory-adjacent process that speeds up the monotony with precision only furthers the efficiency of one’s work. Think of how scouts evaluate players in the modern day, the analytics are fed through stat aggregates with film of specific instances sorted by scripts and algorithms. The time it takes one to grab the data needed to come to their desired conclusion is effortlessly available at the touch of a button; days of sorting and film watching turned into nothing more than an afternoon.

The script created below attempts to achieve these efficient measures, conducting a full on fingerprinting web scan of an IP Address provided and producing a comprehensive set of information.

Contents

[1 Introduction 1](#_Toc194995878)

[1.1 Introduction 1](#_Toc194995879)

[1.2 Aim 1](#_Toc194995880)

[2 Program and Development 3](#_Toc194995881)

[2.1 Program 3](#_Toc194995882)

[2.1.1 Requirements 3](#_Toc194995883)

[2.1.2 Dockerfile 4](#_Toc194995884)

[2.1.3 IP Validation 5](#_Toc194995885)

[2.1.4 Port Scanner 6](#_Toc194995886)

[2.1.5 Banner Grabbing 6](#_Toc194995887)

[2.1.6 Nmap 7](#_Toc194995888)

[2.1.7 Dirb 8](#_Toc194995889)

[2.1.8 Nessus 9](#_Toc194995890)

[2.2 Issues 11](#_Toc194995891)

[2.2.1 Nikto 11](#_Toc194995892)

[2.2.2 Dockerfile Imports 11](#_Toc194995893)

[2.2.3 Socket 11](#_Toc194995894)

[2.2.4 Nessus 11](#_Toc194995895)

[3 Discussion 12](#_Toc194995896)

[3.1 General Discussion 12](#_Toc194995897)

[3.2 Countermeasures 12](#_Toc194995898)

[3.3 Future Work 13](#_Toc194995899)

[References 14](#_Toc194995900)

[Appendices 15](#_Toc194995901)

[Appendix A Dockerfile 15](#_Toc194995902)

[Appendix B Successful Scan Result 15](#_Toc194995903)

[Appendix C – Python Script 17](#_Toc194995904)

# Introduction

## Introduction

The process of gathering unique characteristics and attributes from a computer or network in order to distinguish it from other is how Noah Mainse describes fingerprinting (Mainse, 2023). The process of identifying the properties of the testers entity under examination is crucial in understanding the inner components that could potentially reveal vulnerabilities later. From the port scanning of nmap to the banner grabbing of nikto, the portfolio of the webserver being examined ensures testers are on task with their further examinations. However, the mechanisms of this task are oftentimes repeated throughout different penetration tests. The ability to run several of these programs at once and conduct the fingerprinting process would be immensely useful to a tester.

Python has grown the reputation as the go-to scripting language. Python’s ease of control over OS functions, file manipulation commands, as well as its integration with the command line make it a perfect candidate for manifesting scripts into reality (Pollock 2020). With how python code is written, the developer is essentially writing runnable pseudocode, allowing for a very user-friendly design for code. The limits of python are truly the limits of the programmer utilizing the language, which makes its imports and capabilities perfect for scripting in practice.

The following project is a combination of fingerprint, enumeration, and website penetration testing in the form of a simple python script. The programmer will attempt to create their own script that enables a comprehensive and general test to be run on a website. This will use python scripting language inside a docker on a linux based operating system.

## Aim

The aims of this are to conduct fully comprehensive fingerprinting and enumeration script for a web server that a user provides the IP address for. This project aims to use nmap, dirb, nikto, and Nessus as well as program their very own port scanner and banner grabber to fully output a comprehensive report of the systems. The results of these scans will be outputted into their own text files named according to the scan results they are providing. These scan results will also be displayed to the command line that the user is running the python script through. The programmer aims to achieve this through a working docker container and will provide Dockerfile changes and manipulations to the reader at the end of the paper. To fully immerse the third-party software into the code the programmer plans to use several of the Python import files that exist linked with these services as well as an API service linked to provide the user full functionality of the software.

# Program and Development

## Program

### Requirements

Before the script could be written the programmer utilized several publicly available and accessible python libraries. Each import is utilized for different specific functions and methods within the scripting program. Below is a list of every import the program used:

#### Nmap

Nmap has a publicly available library that allowed its functions to be used seamlessly within the python schema.

#### OS

The os python import allowed for the user to test existing directories as well as add and amend to the directories that exist within the system.

#### Socket

Socket allowed for the user to create a “socket” object that had the capabilities to attempt to ping and connect to web addresses and specific ports.

#### Subprocess

Subprocess allowed for command line commands to be written inside of python through the method “run” which was useful for dirb testing.

#### Requests

Requests printed out the html output of a given web address when prompted with specific parameters. This was useful when programming Nessus API implementation.

#### Json

A Json Python library implementation.

#### Ipaddress

A Python library that ensured that input was in a valid IPv4 format.

#### Rich

Rich was an open-source library that overrides several python commands with “prettier” and more readable through the command line.

A black background with white text

AI-generated content may be incorrect.

Figure 2.1 Imports Required in the script.

### Dockerfile

To ensure that the python script runs the same on all machines a docker container was created before the script was run. This docker had a very own dockerfile that will be provided in Appendix A. From the documentation provided, the python version on docker ran on version 3.8-slim which was a lightweight python variant. Afterwards, the WORKDIR command ensured that the directory in scope was the same for the testing. The apt-get update command was run to ensure all systems were up to date on the machines. The nmap and dirb Linux implementations were both installed with apt-get install -y nmap and apt-get install -y dirb the python imports nmap, rich, and requests were installed after. Finally, the python script was copied onto the docker, and the script ran once the container started up.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 2.2 Screenshot of Dockerfile.

### Command Line Instructions

To ensure the program runs correctly, the programmer recommends opening a command line console in a Linux based operating system. From there, the user should ensure that Docker is installed on the users’ device. The user should navigate to the directory that the program is located within and input the command “sudo docker run -it scantest”. This will run the python script with the assumption that their linux version has docker installed as well as python functionalities.

### IP Validation

In the Figure below there was a code block of the manual IP validation programmed into the script. This would check if the IP input was a valid IVp4 address and would exit out of the program if the input was invalid.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 2.3 Screenshot of IP Validation Method.

### Port Scanner

The first scan conducted by the script was a port scanner manually coded into the script, found in the figure below. This scan checks whether there are any open tcp ports from 0-9999 open on the IP address inputted to the user. For every input, the document will keep track of whether the port is open, filtered, or closed and for open results display them to the user. The open ports are then input into an array and used for later methods inside the testing process.

A computer screen shot of a program

AI-generated content may be incorrect.

2.4 Screenshot of Port Scan Method.

### Banner Grabbing

Using the open ports found in the port scan section, a method attempting to read the information of any of the ports displaying an open webpage was created. This banner grabbing method attempted to find a valid 200 protocol response and displayed the resulting information found from the banner of the webpage. This was done by taking the socket and running the built in recv method that received the data from the webpage. Some of the information displayed by this examination was the connection type, x-content type, http request, date of scan, content-length, and the server’s name.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 2.5 Screenshot of Banner Grab Method.

### Nmap

Utilizing the nmap import the script conducts a complex nmap port scan to confirm the information found previously as well as add to the information gathered. The nmap scan took the -sS -sV and -p- arguments which performed a TCP SYN scan with version detections on all ports. This result was then printed to a file entitled nmapscanresults and these results were printed to the command line being written to the file. This code utilized the structure of the code written in an article by Amal Tom Parakkaden (Parakkaden, 2023).

A computer screen shot of a program code

AI-generated content may be incorrect.

Figure 2.6 Screenshot of Nmap Method.

### Dirb

Since dirb does not have a direct python import implementation, the program utilized the os and subprocess features to properly implement dirb into the script. The script checks for any open ports in the open\_port array from earlier in testing and begins to conduct a dirb scan by checking for the dirb directory existence for the specific target host. Once this directory is created or confirmed by the program the dirb scan writes to a file “dirbscanresults” by using the “subprocess.run” command and testing for webpage extensions within the dirb.txt database. These results were written onto the file and printed into the command line for the user.

A screen shot of a computer code

AI-generated content may be incorrect.

Figure 2.7 Screenshot of dirb method.

### Nessus

The final method inside the program is the implementation of Nessus’ API protocol. This method tried to prompt the user to input their Nessus IP address as well as their accessKey and secretKey. The script would then utilize the Basic Network Scan policy Nessus provided. If the information prompted to the API is confirmed the program will run the test and print out its output to the user. However, incorrect Nessus login information will prompt the user with a “Failed” message with the specific error being printed to the command line as well. This process was heavily influence by a github project done by a user “h3st4k3r” and credit will be given thusly(github, 2025).

A computer screen shot of a program code

AI-generated content may be incorrect.

Figure 2.8 Screenshot of Nessus API method.

## Issues

### Nikto

In the original project proposal, the programmer made the decision that there would be a nikto integration method within the schema. Whether this was an issue with the tester’s environment or the importing process of nikto for whatever reason the Dockerfile was unable to properly import Nikto onto the Docker. There were many solutions purposed, however many with any potential to work such as wiping the machine, factory resetting the Virtual Machine or trying on another machine would have taken the tester far too much time to allocate to. For this reason, nikto was omitted from the final release of the script.

### Dockerfile Imports

Throughout the project there were several instances were figuring out the correct format of the dockerfile ended with several runtime errors. As previously mentioned, nikto alongside several other imports required hours of research to figure out how to properly run in a Docker environment.

### Socket

There were several bugs with socket import implementation in several different methods of the script. Sockets would lead to a runtime error if the object created is not properly closed and destroyed after use. During the programming process, there were several tests where the result would never be fully printed due to the socket object creating a Runtime error. This is an error that was patched upon release of the script.

### Nessus

Nessus API system is very hard to test properly, considering the testers webpage that they are testing on is the locally running Nessus webpage. For this reason, there could be some errors on this section of the program. To mitigate any extending errors from reaching other parts of the script the programmer restricted the Nessus method inside a try method to ensure any errors would be caught and printed regardless of cause or reason. This allows for the remaining sections of the program to run smoothly without any complications from the Nessus API.

# Discussion

## General Discussion

The programmer looked to achieve a full, comprehensive information scan through a python script utilizing several known penetration testing tools and products. From the start, the tester achieved a fully working port scanner, testing open TCP ports on a given IP address as well as a banner grabbing method that was used on the found open ports. The programmer also achieved a fully functional nmap and dirb method, fully utilizing the tools that both imports provided the user with and printing their results into their respective text documents. Finally, the programmer managed to implement the Nessus API and successfully build a fully functional Nessus Basic Network Scan on the IP address given the users Nessus credentials.

The user managed to contain all these results inside of a Dockerfile that has customized imports specifically for the project at hand. This Dockerfile was provided alongside the script to ensure that all users can run the program on their device.

## Countermeasures

Unfortunately, the project did not come without its own issues. The nikto import was unable to function properly within the Docker container. Due to this, there is no method combining the nikto methods with the processes in the script. Furthermore, there were several bugs to do with the socket import as well as formatting within python. The socket import required the tester to close and open a new socket after user, otherwise the program would have a runtime error. On top of this, the Nessus API was a difficult import to work with. The access Key and secret Key variables required were customized long character strings that are difficult to input through a command line. These variables also change very frequently, requiring the user to be aware of their customized API keys regularly.

## Future Work

With more time allotted the tester would work on a nikto import as that is the only aim not met within the scope of this project. The programmer would research possible solutions including possibly downloading a fresh Virtual Machine and coding from the ground up to ensure that no exterior factors are at play in the lack of functionality. The tester also has plans to add several quality-of-life clarity features that showcase the output of the methods clearer through the command line.

# References

**For URLs, Blogs:**

Parakkaden, A. 2023. *Automating Network Scanning with Python and Nmap.* [blog]. 3 May. Available from: <https://medium.com/@amaltomparakkaden/automating-network-scanning-with-python-and-nmap-948948f0b161> [Accessed 4 April 2025].

Bombal, D. 2021. *Github: Red Python Scripts.* [url]. Available from: <https://github.com/davidbombal/red-python-scripts/blob/main/port_scanner_ip_obj.py> [Accessed 4 April 2025].

N.a 2025. *Github: Nessus Tenable API Scripts.* [url]. Available from: <https://github.com/h3st4k3r/nessus-tenable-api-scripts/blob/main/manage-scans.py> [Accessed 5 April 2025].

Mainse, N. 2023. *Understanding Fingerprinting Techniques In Cyber Security.* [blog]. 23 August. Available from: <https://cybermatters.info/cyber-security/understanding-fingerprinting-techniques-in-cyber-security/> [Accessed 5 April 2025].

Pollock, S. 2020. *Why You Should Use Python to Script.* [blog]. 14 July. Available from: <https://medium.com/swlh/why-you-should-use-python-to-script-8f1d591cf2c1> [Accessed 6 April 2025].

# Appendices

## Appendix A Dockerfile

# Use an official lightweight Python image

FROM python:3.8-slim

#FROM kalilinux/kali-rolling

# Set the working directory in the container

WORKDIR /app

# Update and install nmap, dirb imports.

RUN apt-get update && apt-get install -y nmap && apt-get install -y dirb

RUN pip install python-nmap && pip install requests && pip install rich

# Copy the scan script into the container

COPY scan.py .

# Run the scan script when the container starts

CMD ["python", "scan.py"]

#CMD ["sh", "-c", "python nmapscan.py && tail -f /dev/null"]

## Appendix B Successful Scan Result

└─$ sudo docker run -it scantest

IP Address: 192.168.1.253

Initial Port Scan:

[+] Port 8834 is open.

Banner Grab Test:

[+] {port}/tcp is open

[+] banner:

HTTP/1.1 400 Bad Request

X-Content-Type-Options: nosniff

Connection: close

Content-Type: text/html

Date: Fri, 28 Mar 2025 12:27:53 GMT

Content-Length: 349

Server: NessusWWW

Nmap scan results:

Host: 192.168.1.253

State: up

Protocol: tcp

Port: 8834, State: open

Service: nessus-xmlrpc

Product:

Version:

dirb Test:

Scanning target: 192.168.1.253

-----------------

DIRB v2.22

By The Dark Raver

-----------------

OUTPUT\_FILE: reports/dirb/192.168.1.253/dirb.txt

START\_TIME: Fri Mar 28 12:30:28 2025

URL\_BASE: http://192.168.1.253:8834/

WORDLIST\_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

\*\*\* Generating Wordlist...

GENERATED WORDS: 4612

---- Scanning URL: http://192.168.1.253:8834/ ----

\*\*\* Calculating NOT\_FOUND code...

-----------------END\_TIME: Sat Apr 5 09:29:12 2025

DOWNLOADED: 4612 - FOUND: 0

Nessus API Scan Test:

Enter your Nessus IP Addressx

Enter your accessKey: x

Enter your secretKey: x

Please input your Nessus information to get this to work!

## Appendix C – Python Script

###The following program conducts a comprehensive port scan

# on an inputted web server ip address. This web scan utilizes

# nmap, dirb, and various python imports such as socket to find

# open ports and attempt to snoop and search their contents.

# The webserver header, extensions, version, process name, and product ID

# are found and if applicable a nessus web scan is conducted after all this

# information is gathered. Bryce A. Place 2301241@uad.ac.uk

import nmap,os,socket,subprocess,requests,json,ipaddress

from rich import print

# scanner variable that utilizes python nmap import and opens a port scanner.

scanner = nmap.PortScanner()

# prompts user to input IP address

target = input("IP Address: ")

true = True

# try expression to check if the IP address is valid, exits program if invalid.

try:

ipaddress.IPv4Address(target)

true = True

except ipaddress.AddressValueError:

true = False

if not true:

print("Invalid input.")

exit(1)

###open port test

print("Initial Port Scan: ")

# range of potential open ports being examined

ports = range(1,9999)

# open port array

open\_ports = []

# socket variable utilizing socket import, attempts to connect to get a connection with

# the web server. Adds port to open port array if connetion is established.

sock = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

for port in ports:

try:

sock.connect((target,port))

open\_ports.append(port)

except:

pass

sock.close()

# print open port list

for port in open\_ports:

print(f"[+] Port {port} is open.")

###open port test

###banner test

# check if there are any open ports before Banner testing

if (len(open\_ports) > 0):

print("Banner Grab Test: ")

# socket construction

sock = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

for port in open\_ports:

try:

# check if web server is up and giving 200 signal.

# print the banner after tcp port is confirmed open.

sock.connect((target, port))

sock.send(b'200 OK\r\n')

banner = str(sock.recv(256), 'ascii')

print("[+] {port}/tcp is open")

print("[+] banner:")

print(banner)

except socket.error as e:

# if error print if port is filtered or closed whether it times out

# or is invalid.

if 'timed out' in str(e):

error = 'filtered'

else:

error = 'closed'

print("[-] {port}/tcp is {error}")

# close socket regardless

sock.close()

###banner test

###nmap test

print("Nmap Scan Test: ")

# nmap port scanner object, creates an nmap scan with specified parameters.

scanner.scan(target, arguments='-sS -sV -p-')

# write namp scan results into a file, printing the results to the command line.

with open("nmapscanresult.txt", "w") as file:

file.write("Nmap scan results: \n")

# for hosts grabbed by the nmap scan

for host in scanner.all\_hosts():

file.write(f"Host: {host} \n")

file.write(f"State: {scanner[host].state()} \n")

# for protocol in nmap scan

for proto in scanner[host].all\_protocols():

file.write(f"Protocol: {proto}\n")

ports = scanner[host][proto].keys()

# write nmap results into file.

for port in ports:

file.write(f"Port: {port}, State: {scanner[host][proto][port]['state']} \n")

file.write(f"Service: {scanner[host][proto][port]['name']} \n")

file.write(f"Product: {scanner[host][proto][port]['product']} \n")

file.write(f"Version: {scanner[host][proto][port]['version']} \n")

with open("nmapscanresult.txt", "r") as file:

print(file.read())

###nmap test

###dirb test

# confirm there is an open tcp port to test

if (len(open\_ports) > 0):

print("dirb Scan Test: ")

# establish dirb file path

dirb\_dir = f"reports/dirb/{target}"

# establish dirb.txt file

dirb\_file = os.path.join(dirb\_dir, "dirb.txt")

# create directory if doesn't exist

if not os.path.exists(dirb\_dir):

os.makedirs(dirb\_dir)

print(f"Scanning target: {target}")

# write dirb scan results into file

with open("dirbscanresult.txt", "w") as file:

for port in open\_ports:

url = f"http://{target}:{port}"

result = subprocess.run(['dirb', url, '-o', dirb\_file], capture\_output=True, text=True)

file.write(f"{result.stdout}")

# read dirb file scan results

with open("dirbscanresult.txt", "r") as file:

print(file.read())

###dirb test

###nessus test

print("Nessus API Scan Test: ")

try:

# prompt user to input their nessus IP

nessus = input("Enter your Nessus IP Address")

url = f"https://{nessus}:8834"

# commented out akey and skey variables that should be replaced by user if inputting

# keys is far too tedious. Command lines are hard to copy long strings.

#akey = f"input your accessKey"

akey = input("Enter your accessKey: ")

#skey = f"input your secretKey"

skey = input("Enter your secretKey: ")

scans = "/scans"

# nessus API header

headers = {

"X-ApiKeys": f"accessKey={akey}; secretKey={skey}",

"Content-Type": "application/json"

}

# basis web scan data being inputted into the API

data = {

"uuid": "731a8e52-3ea6-a291-ec0a-d2ff0619c19d7bd788d6be818b65",

"settings": {

"name": "scan",

"enabled": True,

"text\_target": f"https://{target}"

}

}

# the actual nessus test request

response = requests.post(f"{url}{scans}", headers=headers, json=data, verify=False)

# prompt user if test was a success, inform user if test failed with reason.

if response.status\_code == 200:

print("[+] Success!")

print("Response: ", response.json())

else:

print("[-] Failed!")

print("Response Status Code:", response.status\_code)

print("Response Text:", response.text)

except:

# if user never inputted nessus IP, prompt user with message.

print("Please input your Nessus information to get this to work!")

###nessus test