Port Scanning with OS Information/Version

1. Quick Overview

1.1 What is Port Scanning?

Port Scanning is a method for determining **Open Ports** in a computer network or on a remote server/host machine. It is usually done by sending connection request to remote server/host machine.

1.2 What is OS Fingerprinting?

OS Fingerprinting is a method for estimating the **Operating System** running on a remote server/host machine. It is usually done by crafting special network packets and analyzing the responses **(Active)** or analyzing the trivial network traffic **(Passive)**.

2. Attack Description

The following programs have been coded and scripts have been written to implement the attack tool:

- `os_fingerprinting.py` (script) is a Python script designed and written for carrying out OS fingerprinting.
- `port_scanning.cpp` (code) is a C++ program designed and coded for carrying out port scanning.
- `run.sh` (script) is a Linux shell script designed and written for compiling and running the attack tool.

The attack sequence and the observed outputs after carrying out an attack are discussed in the following subsections.

2.1 Attack Sequence

Overall attack is carried out in several steps. Each of the programs and scripts mentioned above plays its role in carrying out the attack. The steps of attack are discussed in the following subsections.

2.1.1 Port Scanning with `port_scanning.cpp`

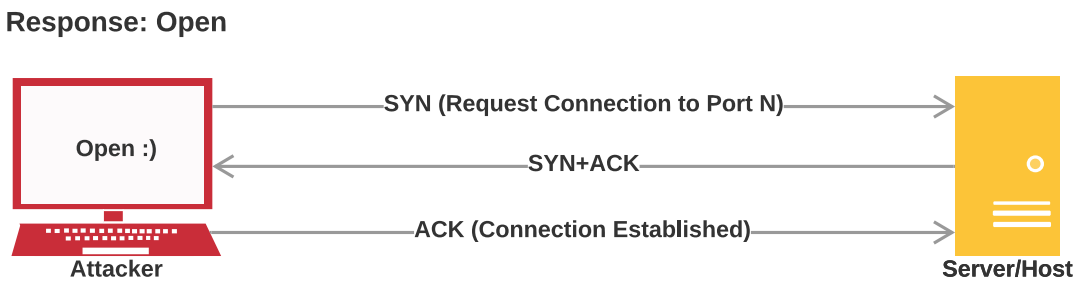
Port scanning is carried out with the following piece of code written in `port_scanning.cpp`.

```
#include<SFML/Network.hpp>

using namespace sf;

bool isPortOpen(const string& ipAddress, int port) {
    return (TcpSocket().connect(ipAddress, port) == Socket::Done);
}
```

Network module of Simple and Fast Multimedia Library (SFML) has been used for this purpose. `isPortOpen()` function takes in a string reference `ipAddress` along with an integer `port` as inputs. Inside this function, a **TCP (Transmission Control Protocol)** connection is established using an instance of TcpSocket class via its `connect()` function. This function takes in the string reference `ipAddress` and converts it to an instance of IpAddress class. After that, it tries to connect the socket to a remote server/host machine with IP address `ipAddress` on port `port`. Then, the returned value from `connect()` function is compared with a predefined `Socket::Done` status code which means the socket has sent/received the data. TCP connection is successfully established with remote server/host machine on specified port, that is, the specified port is open on that machine if `connect()` function returns `Socket::Done`. Finally, the boolean value from the aforementioned comparison is returned from `isPortOpen()` function.



Inside `port_scanning.cpp`, the `main()` function takes in IP address of remote server/host machine along with list of ports to be scanned as string inputs either from console or command line. Then, these inputs are processed with other user-defined functions. Finally, port scanning is carried out targeting the specified ports on remote server/host machine.

2.1.2 OS Fingerprinting with `os_fingerprinting.py`

Inside `main()` function of `port_scanning.cpp`, OS fingerprinting is carried out after port scanning with the following piece of code.

```
system(("sudo python os_fingerprinting.py "+ipAddress).c_str());
```

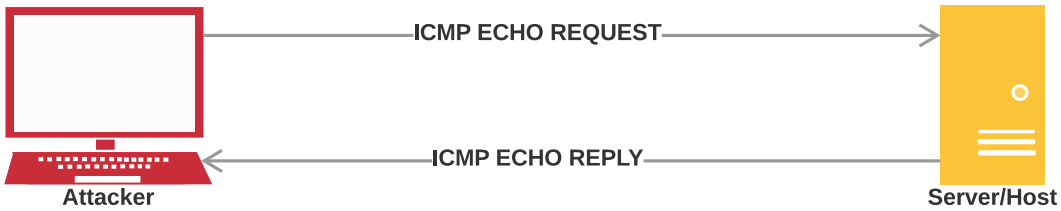
`system()` function is used to invoke an operating system command from a C++ program. So basically, the Python script `os_fingerprinting.py` is invoked and executed from C++ file `port_scanning.cpp`. The IP address of the target server/host machine is provided and the command is invoked with root privilege.

OS fingerprinting is carried out with the following pieces of code written in `os_fingerprinting.py`.

```
from scapy.all import *
from scapy.layers.inet import IP, ICMP

packet = IP(dst=sys.argv[1])/ICMP()
response = sr1(packet, timeout=2, verbose=False)
```

Python module Scapy has been used for this purpose. Scapy is a Python interactive packet manipulation program that enables its users to send, sniff, dissect, and forge network packets. Inside `os_fingerprinting.py` script, `packet` is crafted with **IP (Internet Protocol)** and **ICMP (Internet Control Message Protocol)** layers. IP address of remote server/host machine (`sys.argv[1]`) is set in `dst` field of IP layer. Here, passive OS fingerprinting is carried out with **ICMP echo/Ping messages**. Ping messages are used to find out the availability of a server/host machine on a computer network. After crafting `packet`, ICMP echo request is sent to target server/host machine with `sr1` function. `sr1` function takes in `packet` as one of its inputs and requires root privilege to execute. The function returns ICMP echo reply which is captured in `response`.



```
if response == None:
    print('./{}: No Response from {}.\n'.format(sys.argv[0], sys.argv[1]))
elif IP in response:
    if response.getlayer(IP).ttl <= 64:
        os_guess = 'Linux/ FreeBSD(v5)/ MacOS'
    elif response.getlayer(IP).ttl <= 128:
        os_guess = 'Windows'
    else:
        os_guess = 'Cisco/ Solaris/ SunOS/ FreeBSD(v3.4, v4.0)/ HP-UX(v10.2,
v11)'
```

After capturing ICMP echo reply in `response`, it is analyzed to guess the operating system running on target server/host machine. For this purpose, **TTL (Time to Live) or Hop Count** field of IP layer is examined. TTL is a mechanism which limits the lifespan of data in a computer network. It simply means how long a resolver is supposed to cache the DNS query before the query expires and a new one needs to be done. This TTL value differs among different operating systems which comes in handy while guessing OS running on remote server/host machine. So basically, TTL value of IP layer inside ICMP echo reply is examined to carry out remote OS fingerprinting since this particular TTL value is set by the OS running on remote server/host machine. The default TTL values for different operating systems can be found here.

2.1.3 Running Attack Tool with `run.sh`

The Linux shell script `run.sh` builds raw executable from `port_scanning.cpp` by compilation and linking. Then, it runs the raw executable to carry out port scanning followed by OS fingerprinting.

2.2 Observed Outputs

It should be emphasized that while not explicitly illegal, **Port Scanning and OS Fingerprinting without Permission is Strictly Prohibited**. The owner of the victim server/host machine can sue the person responsible for the attacks. Therefore, all the attacks for testing and reporting purposes have been carried out targeting the following server/host machines:

1. **TOTOLINK Router** with private IP address `192.168.1.1` which is running on **Junos OS (Linux and FreeBSD)**
2. **Laptop** with private IP address `192.168.1.11` which is running on **Windows 10 OS**
3. **Virtual Machine** with private IP address `192.168.1.16` which is running on **Linux OS (Ubuntu 16.04)**
4. `localhost` with IP address `127.0.0.1`
5. `scanme.nmap.org` with IP address `45.33.32.156`

scanme.nmap.org is a service provided by nmap.org and insecure.org. They set up a machine so that enthusiasts can learn about **Nmap (Network Mapper)** and test Nmap installation. The enthusiasts are authorized to carry out port scanning on this machine with Nmap or other port scanners. Not to mention, I own rest of the server/host machines mentioned above.

Following ports have been scanned while carrying out port scanning:

Port Number	Internet Application Running
`port 21`	FTP (File Transfer Protocol)
`port 22`	SSH (Secure Shell Protocol)
`port 23`	Telnet Protocol
`port 24`	Private Mail System
`port 25`	SMTP (Simple Mail Transfer Protocol)
`port 80`	HTTP (Hypertext Transfer Protocol)
`port 443`	HTTPS (HTTP Secure)
`port 8080`	Alternative to `port 80`

With a view to comparing the outputs side by side, all the attacks have been carried out using both the Nmap and the attack tool. The observed outputs from the attacker's perspective **(Virtual Machine with Private IP Address `192.168.1.16`)** are exhibited in the following subsections.

2.2.1 Victim: `192.168.1.1`

Nmap

```
fromsaffroncity@fromsaffroncity:~$ sudo nmap -O -p 21-25,80,443,8080 192.168.1.1

Starting Nmap 7.01 ( https://nmap.org ) at 2021-07-22 11:11 +06
Nmap scan report for my.totolink.com (192.168.1.1)
Host is up (0.0056s latency).
PORT     STATE  SERVICE
21/tcp   closed ftp
22/tcp   closed ssh
23/tcp   closed telnet
24/tcp   closed priv-mail
25/tcp   closed smtp
80/tcp   open   http
443/tcp  closed https
8080/tcp closed http-proxy
MAC Address: 78:44:76:A2:CD:4C (Zioncom technology)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.9 - 2.6.33
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 2.40 seconds
```

Attack Tool

```
fromsaffroncity@fromsaffroncity:~/Desktop/cse406/project-src/main$ ./run.sh 192.168.1.1 21-25,80,443,8080

----------------------------------------
|    Port Scanner & OS Version Guesser    |
----------------------------------------

./port.scanner: Port Scanning (192.168.1.1)...

Port   80: OPEN

./port.scanner: Port Scanning Completed, 1/8 ports OPEN.

WARNING: No route found for IPv6 destination :: (no default route?)

./os_fingerprinting.py: OS Fingerprinting (192.168.1.1)...

./os_fingerprinting.py: TTL = 64 -> Guessed OS = Linux/ FreeBSD(v5)/ MacOS.

----------------------------------------
|             Thank You !!!              |
----------------------------------------
```

By observing the outputs from both the Nmap and the attack tool for IP address `192.168.1.1` and ports `21-25, 80, 443, 8080`, it can be concluded that target server/host machine is listening on `port 80` for HTTP connection. Also, a version of Linux OS may be running on the target machine.

2.2.2 Victim: `192.168.1.11`

Nmap

```
fromsaffroncity@fromsaffroncity:~$ sudo nmap -O -p 21-25,80,443 192.168.1.11

Starting Nmap 7.01 ( https://nmap.org ) at 2021-07-22 11:15 +06
Nmap scan report for 192.168.1.11
Host is up (0.00017s latency).
PORT    STATE    SERVICE
21/tcp  filtered ftp
22/tcp  filtered ssh
23/tcp  filtered telnet
24/tcp  filtered priv-mail
25/tcp  filtered smtp
80/tcp  filtered http
443/tcp filtered https
MAC Address: F0:03:8C:A2:AC:BF (Unknown)
Too many fingerprints match this host to give specific OS details
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 17.20 seconds
```

Attack Tool

```
fromsaffroncity@fromsaffroncity:~/Desktop/cse406/project-src/main$ ./run.sh 192.168.1.11 21-25,80,443

-----------------------------------------
|    Port Scanner & OS Version Guesser    |
-----------------------------------------

./port.scanner: Port Scanning (192.168.1.11)...

./port.scanner: Port Scanning Completed, 0/7 ports OPEN.

WARNING: No route found for IPv6 destination :: (no default route?)

./os_fingerprinting.py: OS Fingerprinting (192.168.1.11)...

./os_fingerprinting.py: TTL = 127 -> Guessed OS = Windows.

-----------------------------------------
|              Thank You !!!              |
-----------------------------------------
```

By observing the outputs from both the Nmap and the attack tool for IP address `192.168.1.11` and ports `21-25, 80, 443`, it can be concluded that target server/host machine is not listening on any port for new connection. Also, Nmap has failed to guess the exact OS running on the target machine as an excessive number of existing OS fingerprints matched the signatures. On the other hand, attack tool has guessed that a version of Windows OS may be running on the target machine by analyzing the TTL value.

2.2.3 Victim: `192.168.1.16`

Nmap

```
fromsaffroncity@fromsaffroncity:~$ sudo nmap -O -p 21-25,80,443 192.168.1.16

Starting Nmap 7.01 ( https://nmap.org ) at 2021-07-22 11:15 +06
Nmap scan report for 192.168.1.16
Host is up (0.000044s latency).
PORT    STATE  SERVICE
21/tcp  closed ftp
22/tcp  open   ssh
23/tcp  closed telnet
24/tcp  closed priv-mail
25/tcp  closed smtp
80/tcp  closed http
443/tcp closed https
Device type: general purpose
Running: Linux 3.X
OS CPE: cpe:/o:linux:linux_kernel:3
OS details: Linux 3.8 - 3.19
Network Distance: 0 hops

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 15.58 seconds
```

Attack Tool

```
fromsaffroncity@fromsaffroncity:~/Desktop/cse406/project-src/main$ ./run.sh 192.168.1.16 21-25,80,443

----------------------------------------
|    Port Scanner & OS Version Guesser    |
----------------------------------------

./port.scanner: Port Scanning (192.168.1.16)...

Port  22: OPEN

./port.scanner: Port Scanning Completed, 1/7 ports OPEN.

WARNING: No route found for IPv6 destination :: (no default route?)

./os_fingerprinting.py: OS Fingerprinting (192.168.1.16)...

./os_fingerprinting.py: No Response from 192.168.1.16.

----------------------------------------
|             Thank You !!!             |
----------------------------------------
```

By observing the outputs from both the Nmap and the attack tool for IP address `192.168.1.16` and ports `21-25, 80, 443`, it can be concluded that target server/host machine is listening on `port 22` for SSH connection. Also, Nmap has guessed that a version of Linux OS may be running on the target machine. On the other hand, attack tool has failed to guess the OS running on the target machine as remote server/host machine did not respond to ICMP echo request.

2.2.4 Victim: `localhost`

Nmap

```
fromsaffroncity@fromsaffroncity:~$ sudo nmap -O -p 21-25,80,443 localhost

Starting Nmap 7.01 ( https://nmap.org ) at 2021-07-22 11:17 +06
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000021s latency).
PORT    STATE  SERVICE
21/tcp  closed ftp
22/tcp  open   ssh
23/tcp  closed telnet
24/tcp  closed priv-mail
25/tcp  closed smtp
80/tcp  closed http
443/tcp closed https
Device type: general purpose
Running: Linux 3.X
OS CPE: cpe:/o:linux:linux_kernel:3
OS details: Linux 3.12 - 3.19, Linux 3.8 - 3.19
Network Distance: 0 hops

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 2.56 seconds
```

Attack Tool

```
fromsaffroncity@fromsaffroncity:~/Desktop/cse406/project-src/main$ ./run.sh localhost 21-25,80,443

----------------------------------------
|    Port Scanner & OS Version Guesser    |
----------------------------------------

./port.scanner: Port Scanning (localhost)...

Port  22: OPEN

./port.scanner: Port Scanning Completed, 1/7 ports OPEN.

WARNING: No route found for IPv6 destination :: (no default route?)

./os_fingerprinting.py: OS Fingerprinting (localhost)...

./os_fingerprinting.py: No Response from localhost.

----------------------------------------
|             Thank You !!!             |
----------------------------------------
```

By observing the outputs from both the Nmap and the attack tool for `localhost` and ports `21-25, 80, 443`, it can be concluded that target server/host machine is listening on `port 22` for SSH connection. Also, Nmap has guessed that a version of Linux OS may be running on the target machine. On the other hand, attack tool has failed to guess the OS running on the target machine as remote server/host machine did not respond to ICMP echo request.

2.2.5 Victim: `scanme.nmap.org`

Nmap

```
fromsaffroncity@fromsaffroncity:~$ sudo nmap -O -p 21-25,80,443 scanme.nmap.org

Starting Nmap 7.01 ( https://nmap.org ) at 2021-07-22 11:18 +06
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.26s latency).
Other addresses for scanme.nmap.org (not scanned): 2600:3c01::f03c:91ff:fe18:bb2f
PORT    STATE  SERVICE
21/tcp  closed ftp
22/tcp  open   ssh
23/tcp  closed telnet
24/tcp  closed priv-mail
25/tcp  closed smtp
80/tcp  open   http
443/tcp closed https
Aggressive OS guesses: HP P2000 G3 NAS device (93%), Linux 3.1 (91%), Linux 3.2 (91%), AXIS 210A or 211 Network Camera (Linux 2.6.17) (91%), Asus RT-AC66U rou
ter (Linux 2.6) (90%), Asus RT-N16 WAP (Linux 2.6) (90%), Asus RT-N66U WAP (Linux 2.6) (90%), Tomato 1.28 (Linux 2.6.22) (90%), Linux 2.6.18 - 2.6.22 (90%), O
penWrt 0.9 - 7.09 (Linux 2.4.30 - 2.4.34) (90%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 16 hops

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 10.45 seconds
```

Attack Tool

```
fromsaffroncity@fromsaffroncity:~/Desktop/cse406/project-src/main$ ./run.sh scanme.nmap.org 21-25,80,443

-----------------------------------------
|    Port Scanner & OS Version Guesser    |
-----------------------------------------

./port.scanner: Port Scanning (scanme.nmap.org)...

Port  22: OPEN
Port  80: OPEN

./port.scanner: Port Scanning Completed, 2/7 ports OPEN.

WARNING: No route found for IPv6 destination :: (no default route?)

./os_fingerprinting.py: OS Fingerprinting (scanme.nmap.org)...

./os_fingerprinting.py: TTL = 47 -> Guessed OS = Linux/ FreeBSD(v5)/ MacOS.

-----------------------------------------
|              Thank You !!!              |
-----------------------------------------
```

By observing the outputs from both the Nmap and the attack tool for `scanme.nmap.org` and ports `21-25, 80, 443`, it can be concluded that target server/host machine is listening on `port 22` and `port 80` for SSH and HTTP connections respectively. Also, Nmap has failed precisely guess the OS running on the target machine as no existing OS fingerprint exactly matched the signatures. On the other hand, attack tool has guessed that a version of Linux OS may be running on the target machine by analyzing the TTL value.

3. Attack Analysis

The concerned attack tool carries out two individual attacks. These are:

1. Port Scanning
2. OS Fingerprinting

Here, outputs from the sophisticated network mapper Nmap has also been observed alongside outputs from the attack tool. After comparing and analyzing their performances, it can be concluded that the attack tool has successfully carried out the port scanning on the remote server/host machine but its performance in OS fingerprinting has not been up to the mark.

Port scanning involves establishment of TCP connection with remote machine through three-way handshaking. This can be easily done with sockets. Hence, the attack tool has not faced any difficulties in carrying out port scanning and its performance has been up to the mark.

On the other hand, OS fingerprinting involves sophisticated process of crafting special network packets (active) or analyzing and comparing network traffic for finding out key signatures in TCP/IP stack implementation of remote server/host machine (passive). Nmap carries out a delicate version of active OS fingerprinting. Contrastingly, the attack tool carries out passive OS fingerprinting through Ping messages with only TTL value in consideration. In order to carry out a successful passive OS fingerprinting, other signatures of TCP/IP stack implementation should also be considered and different test should also be run. This is where the attack tool lacks and therefore, its performance has not been up to the mark. However, it has guessed the remote OS correctly for some cases.

Overall, the attacks carried out can be considered successful to a considerable extent.

4. OS Fingerprinting with Available Tool

In order to abide by the imposed constraints and enhance the performance, OS fingerprinting tool available on the Github repository OS-Fingerprinting (coded and written by Cesar Ghali) has been thoroughly studied and examined.

This tool carries out passive OS fingerprinting where legitimate computer network traffic is analyzed and compared for certain key differences in the **TCP/IP Stack Implementation** on different types and versions of operating systems. It sends a legitimate **HTTP (Hypertext Transfer Protocol)** request to the victim server/host machine. Then, it runs different tests, based on the response packets, to estimate the type and version of installed OS on the victim machine. This tool takes into account 10 different TCP/IP stack features to filter out the final estimation from 24 different operating system types and versions. These features have different weight values based on their accuracy.

The main reason behind choosing this particular tool for further examination has been its usage of computer networking libraries and utilities provided in C language. Contrastingly, the attack tool I have designed implements OS fingerprinting with Python language and its networking utilities.

After thorough exploration and examination, OS fingerprinting has been carried out using the concerned tool. The observed outputs from the attacker's perspective **(Virtual Machine with Private IP Address `192.168.1.16`)** are exhibited in the following subsections.

4.1 Victim: `192.168.1.1.`

```
fromsaffroncity@fromsaffroncity:~/Desktop/cse406/OS-Fingerprinting/Attack$ sudo ./ofp_tool 192.168.1.1


Initializing the packet sniffer...
OS fingerprinting the network device [192.168.1.1] through port [80]...

-> Operating Systems probability weight:
   -----------------------------------
---> Linux 2.1                       1.5
---> Linux 2.0                       1.5
---> Linux 2.0.3x                    1.5
---> Linux 2.2                       7.0
---> Linux 2.4                       5.0
---> Linux 2.6                       6.0
---> Windows 3.11                    2.0
---> Windows 95                      3.5
---> Windows 95b                     3.0
---> Windows 98                      2.0
---> Windows ME no SP                1.5
---> Windows NT 4.0 SP6a             2.0
---> Windows 2000 SP2+               2.0
---> Windows 2000 SP3                2.0
---> Windows 2000 SP4                2.0
---> Windows XP SP1+                 2.0
---> Windows 2K3                     1.5
---> Windows Vista (beta)            2.0
---> MacOS 7.3-8.6 (OTTCP)           2.0
---> MacOS 8.1-8.6 (OTTCP)           2.0
---> MacOS 8.6                       2.0
---> MacOS 9.0-9.2                   2.0
---> MacOS 9.1 (OT 2.7.4)            2.0
---> MacOS 10.2.6                    3.5

-> Possible Operating Systems:
------------------------------
---> Linux 2.2
---> Running Web Server: Boa/0.94.14rc21
```

The observed output align with the estimations made by both the Nmap and the attack tool.

4.2 Victim: `45.33.32.156`

```
fromsaffroncity@fromsaffroncity:~/Desktop/cse406/OS-Fingerprinting/Attack$ sudo ./ofp_tool 45.33.32.156


Initializing the packet sniffer...
OS fingerprinting the network device [45.33.32.156] through port [80]...

-> Operating Systems probability weight:
   -----------------------------------
---> Linux 2.1                       1.5
---> Linux 2.0                       1.5
---> Linux 2.0.3x                    1.5
---> Linux 2.2                       7.0
---> Linux 2.4                       5.0
---> Linux 2.6                       6.0
---> Windows 3.11                    2.0
---> Windows 95                      3.5
---> Windows 95b                     3.0
---> Windows 98                      2.0
---> Windows ME no SP                1.5
---> Windows NT 4.0 SP6a             2.0
---> Windows 2000 SP2+               2.0
---> Windows 2000 SP3                2.0
---> Windows 2000 SP4                2.0
---> Windows XP SP1+                 2.0
---> Windows 2K3                     1.5
---> Windows Vista (beta)            2.0
---> MacOS 7.3-8.6 (OTTCP)           2.0
---> MacOS 8.1-8.6 (OTTCP)           2.0
---> MacOS 8.6                       2.0
---> MacOS 9.0-9.2                   2.0
---> MacOS 9.1 (OT 2.7.4)            2.0
---> MacOS 10.2.6                    3.5

-> Possible Operating Systems:
------------------------------
---> Linux 2.2
---> Running Web Server: Apache/2.4.7 (Ubuntu)
```

The observed output align with the estimations made by both the Nmap and the attack tool.

5. Countermeasures

A complete solution for preventing port scanning as well as OS fingerprinting is impossible to a certain degree. This is mainly because many systems on computer networks have ports open and listening to new connection for certain applications. This is particularly applicable for web servers. In addition to that, new methods for carrying out these attacks are designed and developed on a regular basis to evade the state of the art detection mechanisms.

Nevertheless, a good number of countermeasures for port scanning and OS fingerprinting have been designed and developed over the time. These countermeasures are discussed in the following subsections.

5.1 Countermeasures for Port Scanning

It is equally applicable in the filed of computer security that the best offence is a good defense. As long as there is a publicly accessible server/host machine, a computer network system will be vulnerable to port scanning attack. But, there are handful of things to do for limiting its vulnerabilities.

5.1.1 Firewall

By installing a firewall, unauthorized access to private computer network can be prevented. It manages the ports that are publicly exposed and visible. Firewalls can also detect a port scanning attack in progress and shut the malicious connection down.

5.1.2 TCP Wrapper

TCP wrapper can provide network administrators the flexibility to permit or deny access to the server/host machines based on IP address as well as domain name.

5.1.3 Self Port Scanning

In order to uncover holes in the computer network, internal port scanning can be conducted to determine if there are more open ports than required. Server/host machine can be checked periodically to determine existing vulnerable points that can be exploited.

5.2 Countermeasures for OS Fingerprinting

In order to protect server/host machines on a computer network from OS fingerprinting, there are multiple ways to do so.

Definitely, carrying out active and passive OS fingerprinting on server/host machines registered in the concerned network is a must. In this way, the information that an attacker may get by doing the same on the machines can be discovered.

Also, properly configured, implemented, and maintained **IDS (Intrusion Detection System)**, **IPS (Intrusion Prevention System)**, and **Firewall** can mitigate active OS fingerprinting.

On the other hand, passive OS fingerprinting can be mitigated by ensuring that **NIC (Network Interface Card)** do not operate in promiscuous mode. If some NICs must operate promiscuously for the sake of proper functionality, then they should be monitored closely on a regular basis.

In addition to that, usage of hubs in computer network should be avoided. Instead, properly configured switches can be used.

Implementing strong encryption as much as possible in a computer network also makes packet sniffing difficult for the attackers.

And finally, all of the network logs should be checked frequently. Often, many network attacks can be prevented by analyzing logs on a regular basis.