

Username: Palm Beach State College IP Holder **Book:** Kali Linux – Assuring Security by Penetration Testing. No part of any chapter or book may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher of the book or chapter. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates these Terms of Service is strictly prohibited. Violators will be prosecuted to the full extent of U.S. Federal and Massachusetts laws.

Chapter 5. Target Discovery

In this chapter, we will describe the process of discovering machines on the target network using various tools available in Kali Linux. We will explain the following topics:

- A description of the target discovery process
- The method used to identify target machines using the tools in Kali Linux
- The steps required to find the operating systems of the target machines (operating system fingerprinting)

To help you understand these concepts easily, we will use a virtual network as the target network.

Starting off with target discovery

After we have gathered information about our target network from third-party sources, such as search engines, the next step would be to discover our target machines. The purpose of this process is as follows:

- To find out which machine in the target network is available. If the target machine is not available, we won't continue the penetration testing process on that machine and move to the next machine.
- To find the underlying operating system used by the target machine.

Collecting the previously mentioned information will help us during the vulnerabilities mapping process.

We can utilize the tools provided in Kali Linux for the target discovery process. Most of these tools are available in the **Information Gathering** menu, with the following submenus:

- **Identify Live Hosts**
- **OS Fingerprinting**

In this chapter, we will only describe a few important tools in each category. The tools are selected based on the functionality, popularity, and the tool development activity.

Username: Palm Beach State College IP Holder **Book:** Kali Linux – Assuring Security by Penetration Testing. No part of any chapter or book may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher of the book or chapter. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates these Terms of Service is strictly prohibited. Violators will be prosecuted to the full extent of U.S. Federal and Massachusetts laws.

Identifying the target machine

The tools included in this category are used to identify the target machines that can be accessed by a penetration tester. Before we start the identification process, we need to know our client's terms and agreements. If the agreements require us to hide pen-testing activities, we need to conceal our penetration testing activities. Stealth technique may also be applied for testing the **Intrusion Detection System (IDS)** or **Intrusion Prevention System (IPS)** functionality. If there are no such requirements, we may not need to conceal our penetration testing activities.

ping

The **ping** tool is the most famous tool that is used to check whether a particular host is available. The **ping** tool works by sending an **Internet Control Message Protocol (ICMP)** echo request packet to the target host. If the target host is available and the firewall is not blocking the ICMP echo request packet, it will reply with the ICMP echo reply packet.

Note

The ICMP echo request and ICMP echo reply are two of the available ICMP control messages. For other ICMP control messages, you can refer to the following URL:

https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol#Control_messages

Although you can't find **ping** in the Kali Linux menu, you can open the console and type the **ping** command with its options.

To use **ping**, you can just type **ping** and the destination address as shown in the following screenshot:

```
root@kali:~# ping 192.168.56.102
PING 192.168.56.102 (192.168.56.102) 56(84) bytes of data.
64 bytes from 192.168.56.102: icmp_req=1 ttl=64 time=1.03 ms
64 bytes from 192.168.56.102: icmp_req=2 ttl=64 time=0.421 ms
64 bytes from 192.168.56.102: icmp_req=3 ttl=64 time=0.428 ms
64 bytes from 192.168.56.102: icmp_req=4 ttl=64 time=0.503 ms
64 bytes from 192.168.56.102: icmp_req=5 ttl=64 time=0.510 ms
64 bytes from 192.168.56.102: icmp_req=6 ttl=64 time=0.741 ms
64 bytes from 192.168.56.102: icmp_req=7 ttl=64 time=0.503 ms
64 bytes from 192.168.56.102: icmp_req=8 ttl=64 time=0.771 ms
64 bytes from 192.168.56.102: icmp_req=9 ttl=64 time=0.477 ms
64 bytes from 192.168.56.102: icmp_req=10 ttl=64 time=0.522 ms
^C
--- 192.168.56.102 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9000ms
rtt min/avg/max/mdev = 0.421/0.590/1.033/0.188 ms
```

In Kali Linux, by default, **ping** will run continuously until you press **Ctrl + C**.

The **ping** tool has a lot of options, but the following are a few options that are often used:

- The **-C** count: This is the number of echo request packets to be sent.
- The **-I** interface address: This is the network interface of the source address. The argument may be a numeric IP address (such as **192.168.56.102**) or the name of the device (such as **eth0**). This option is required if you want to ping the IPv6 link-local address.
- The **-S** packet size: This specifies the number of data bytes to be sent. The default is 56 bytes, which translates into 64 ICMP data bytes when combined with the 8 bytes of the ICMP header data.

Let's use the preceding information in practice.

Suppose you are starting with internal penetration testing work. The customer gave you access to their network using a LAN cable. And, they also gave you the list of target servers' IP addresses.

The first thing you would want to do before launching a full penetration testing arsenal is to check whether these servers are accessible from your machine. You can use `ping` for this task.

The target server is located at `192.168.56.102`, while your machine has an IP address of `192.168.56.101`. To check the target server availability, you can give the following command:

```
ping -c 1 192.168.56.102
```

Note

Besides IP addresses, `ping` also accepts hostnames as the destination.

The following screenshot is the result of the preceding `ping` command:

```
root@kali:~# ping -c 1 192.168.56.102
PING 192.168.56.102 (192.168.56.102) 56(84) bytes of data.
64 bytes from 192.168.56.102: icmp_req=1 ttl=64 time=1.32 ms

--- 192.168.56.102 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.326/1.326/1.326/0.000 ms
```

From the preceding screenshot, we know that there is one ICMP echo request packet sent to the destination (IP address: `192.168.56.102`). Also, the sending host (IP address: `192.168.56.101`) received one ICMP echo reply packet. The round-trip time required is **1.326 ms**, and there is no packet loss during the process.

Let's see the network packets that are transmitted and received by our machine. We are going to use **Wireshark**, a network protocol analyzer, on our machine to capture these packets, as shown in the following screenshot:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.56.101	192.168.56.102	ICMP	98	Echo (ping) request id=0x0c78, seq=1/256, ttl=64
2	0.004454000	192.168.56.102	192.168.56.101	ICMP	98	Echo (ping) reply id=0x0c78, seq=1/256, ttl=64

From the preceding screenshot, we can see that our host (`192.168.56.101`) sent one ICMP echo request packet to the destination host (`192.168.56.102`). Since the destination is alive and allows the ICMP echo request packet, it will send the ICMP echo reply packet back to our machine.

Note

We will cover Wireshark in more detail in the *Network sniffers* section in [Chapter 10, Privilege Escalation](#).

If your target is using an IPv6 address, such as `fe80::a00:27ff:fe43:1518`, you can use the `ping6` tool to check its availability. You need to give the `-I` option for the command to work against the link-local address:

```
# ping6 -c 1 fe80::a00:27ff:fe43:1518 -I eth0
PING fe80::a00:27ff:fe43:1518(fe80::a00:27ff:fe43:1518) from
fe80::a00:27ff:fe1c:5122 eth0: 56 data bytes
64 bytes from fe80::a00:27ff:fe43:1518: icmp_seq=1 ttl=64 time=4.63 ms

--- fe80::a00:27ff:fe43:1518 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 4.633/4.633/4.633/0.000 ms
```

The following screenshot shows the packets sent to complete the `ping6` request:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::a00:27ff:fe1c:5122	ff02::1:ff43:1518	ICMPv6	86	Neighbor Solicitation for fe80::a00:27ff:fe43:1518 from 08:00:27:1c:51:22
2	0.001852000	fe80::a00:27ff:fe43:1518	fe80::a00:27ff:fe1c:5122	ICMPv6	86	Neighbor Advertisement fe80::a00:27ff:fe43:1518 (sol, ovr) is at 08:00:27:43:15:18
3	0.001933000	fe80::a00:27ff:fe1c:5122	fe80::a00:27ff:fe43:1518	ICMPv6	118	Echo (ping) request id=0x0d16, seq=1
4	0.004551000	fe80::a00:27ff:fe43:1518	fe80::a00:27ff:fe1c:5122	ICMPv6	118	Echo (ping) reply id=0x0d16, seq=1
5	5.012092000	fe80::a00:27ff:fe43:1518	fe80::a00:27ff:fe1c:5122	ICMPv6	86	Neighbor Solicitation for fe80::a00:27ff:fe1c:5122 from 08:00:27:43:15:18
6	5.012167000	fe80::a00:27ff:fe1c:5122	fe80::a00:27ff:fe43:1518	ICMPv6	78	Neighbor Advertisement fe80::a00:27ff:fe1c:5122 (sol)

From the preceding screenshot, we know that `ping6` is using the `ICMPv6` request and reply.

To block the `ping` request, the firewall can be configured to only allow the ICMP echo request packet from a specific host and drop the packets sent from other hosts.

arping

The `arping` tool is used to ping a host in the **Local Area Network (LAN)** using the **Address Resolution Protocol (ARP)** request. You can use `arping` to ping a target machine using its IP, host, or **Media Access Control (MAC)** address.

The `arping` tool operates on **Open System Interconnection (OSI)** layer 2 (network layer), and it can only be used in a local network. Moreover, ARP cannot be routed across routers or gateways.

To start `arping`, you can use the console to execute the following command:

```
# arping
```

This will display brief usage information on `arping`.

You can use `arping` to get the target host's MAC address:

```
# arping 192.168.56.102 -c 1
ARPING 192.168.56.102
60 bytes from 08:00:27:43:15:18 (192.168.56.102): index=0 time=518.223 usec

--- 192.168.56.102 statistics ---
1 packets transmitted, 1 packets received, 0% unanswered (0 extra)
```

From the previous command output, we can see that the target machine has a MAC address of `08:00:27:43:15:18`.

Let's observe the network packets captured by Wireshark on our machine during the `arping` process:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	08:00:27:1c:51:22	ff:ff:ff:ff:ff:ff	ARP	42	Who has 192.168.56.102? Tell 192.168.56.101
2	0.001643000	08:00:27:43:15:18	08:00:27:1c:51:22	ARP	60	192.168.56.102 is at 08:00:27:43:15:18

From the preceding screenshot, we can see that our network card (MAC address: `08:00:27:1c:51:22`) sends an ARP request to a broadcast MAC address (`ff:ff:ff:ff:ff:ff`), looking for the IP address `192.168.56.102`. If the IP address `192.168.56.102` exists, it will send an ARP reply mentioning its MAC address (`08:00:27:43:15:18`), as can be seen from packet number `2`.

However, if the IP address is not available, there will be no ARP replies, informing the MAC address of the `192.168.56.103` IP address, as can be seen from the following screenshot:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	08:00:27:1c:51:22	ff:ff:ff:ff:ff:ff	ARP	42	Who has 192.168.56.103? Tell 192.168.56.101
2	1.002377000	08:00:27:1c:51:22	ff:ff:ff:ff:ff:ff	ARP	42	Who has 192.168.56.103? Tell 192.168.56.101

Another common use of `arping` is to detect duplicate IP addresses in a local network. For example, your machine is usually connected to a local network using an IP address of `192.168.56.101`; one day, you would like to change the IP address. Before you can use the new IP address, you need to check whether that particular IP address has already been used.

You can use the following `arping` command to help you detect whether the IP address of `192.168.56.102` has been used:

```
# arping -d -i eth0 192.168.56.102 -c 2
# echo $?
1
```

If the code returns `1`, it means that the IP address of `192.168.56.102` has been used by more than one machine. Whereas, if the code returns `0`, it means that the IP address is available.

fping

The difference between `ping` and `fping` is that the `fping` tool can be used to send a ping (ICMP echo) request to several hosts at once. You can specify several targets on the command line, or you can use a file containing the hosts to be pinged.

In the default mode, `fping` works by monitoring the reply from the target host. If the target host sends a reply, it will be noted and removed from the target list. If the host doesn't respond for a certain time limit, it will be marked as `unreachable`. By default, `fping` will try to send three ICMP echo request packets to each target.

To access `fping`, you can use the console to execute the following command:

```
# fping -h
```

This will display the description of usage and options available in `fping`.

The following scenarios will give you an idea of the `fping` usage:

- If we want to know the alive hosts of `192.168.1.1`, `192.168.1.100` and `192.168.1.107` at once, we can use the following command:

```
fping 192.168.1.1 192.168.1.100 192.168.1.107
```

The following is the result of the preceding command:

```
192.168.1.1 is alive
192.168.1.107 is alive
ICMP Host Unreachable from 192.168.1.112 for ICMP Echo sent to
192.168.1.100
ICMP Host Unreachable from 192.168.1.112 for ICMP Echo sent to
192.168.1.100
ICMP Host Unreachable from 192.168.1.112 for ICMP Echo sent to
192.168.1.100
192.168.1.100 is unreachable
```

- We can also generate the host list automatically without defining the IP addresses one by one and identifying the alive hosts. Let's suppose we want to know the alive hosts in the `192.168.56.0` network; we can use the `-g` option and define the network to check, using the following command:

```
# fping -g 192.168.56.0/24
```

The result for the preceding command is as follows:

```
192.168.56.101 is alive
192.168.56.102 is alive
ICMP Host Unreachable from 192.168.56.102 for ICMP Echo sent to
192.168.56.2
ICMP Host Unreachable from 192.168.56.102 for ICMP Echo sent to
192.168.56.3
ICMP Host Unreachable from 192.168.56.102 for ICMP Echo sent to
192.168.56.4
ICMP Host Unreachable from 192.168.56.102 for ICMP Echo sent to
192.168.56.5
ICMP Host Unreachable from 192.168.56.102 for ICMP Echo sent to
192.168.56.6
...
192.168.56.252 is unreachable
192.168.56.253 is unreachable
192.168.56.254 is unreachable
```

- If we want to change the number of ping attempts made to the target, we can use the `-r` option (retry limit) as shown in the following command line. By default, the number of ping attempts is three.

```
fping -r 1 -g 192.168.1.1 192.168.1.10
```

The result of the command is as follows:

```
192.168.1.1 is alive
192.168.1.10 is alive
192.168.1.2 is unreachable
...
192.168.1.9 is unreachable
```

- Displaying the cumulative statistics can be done by giving the `-S` option (print cumulative statistics) as follows:

```
fping -s www.yahoo.com www.google.com www.msn.com
```

The following is the result of the preceding command line:

```
www.google.com is alive
www.yahoo.com is alive
www.msn.com is unreachable
    3 targets
    2 alive
    1 unreachable
    0 unknown addresses
    4 timeouts (waiting for response)
    6 ICMP Echos sent
    2 ICMP Echo Replies received
    0 other ICMP received
51.6 ms (min round trip time)
231 ms (avg round trip time)
411 ms (max round trip time)
4.150 sec (elapsed real time)
```

hping3

The `hping3` tool is a command-line network packet generator and analyzer tool. The capability to create custom network packets allows `hping3` to be used for TCP/IP and security testing, such as port scanning, firewall rule testing, and network performance testing.

The following are several other uses of `hping3` according to the developer (<http://wiki.hping.org/25>):

- Test firewall rules
- Test **Intrusion Detection System (IDS)**
- Exploit known vulnerabilities in the TCP/IP stack

To access `hping3` , go to the console and type `hping3` .

You can give commands to `hping3` in several ways, via the command line, interactive shell, or script.

Without any given command-line options, `hping3` will send a null TCP packet to port `0` .

In order to change to a different protocol, you can use the following options in the command line to define the protocol:

No.	Short option	Long option	Description
-----	--------------	-------------	-------------

No.	Short option	Long option	Description
1	-0	--raw-ip	This sends raw IP packets
2	-1	--icmp	This sends ICMP packets
3	-2	--udp	This sends UDP packets
4	-8	--scan	This indicates the scan mode
5	-9	--listen	This indicates the listen mode

When using the TCP protocol, we can use the TCP packet without any flags (this is the default behavior) or we can give one of the following flag options:

No.	Option	Flag name
1	-S	syn
2	-A	ack
3	-R	rst
4	-F	fin
5	-P	psh
6	-U	urg
7	-X	xmas: flags fin, urg, psh set
8	-Y	ymas

Let's use **hping3** for several cases as follows:

- Send one ICMP echo request packet to a **192.168.56.101** machine. The options used are **-1** (for the ICMP protocol) and **-c 1** (to set the count to one packet):

```
hping3 -1 192.168.56.101 -c 1
```

The following is the output of the command:

```

root@kali:~# hping3 -1 192.168.56.101 -c 1
HPING 192.168.56.101 (eth0 192.168.56.101): icmp mode set, 28 headers + 0 data bytes
len=46 ip=192.168.56.101 ttl=64 id=33099 icmp_seq=0 rtt=9.0 ms

--- 192.168.56.101 hping statistic ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 9.0/9.0/9.0 ms

```

From the preceding output, we can note that the target machine is alive because it has replied to our ICMP echo request.

To verify this, we captured the traffic using `tcpdump` and the following screenshot shows the packets:

```

20:23:04.411622 IP 192.168.56.102 > 192.168.56.101: ICMP echo request, id 7182, seq 0, length 8
20:23:04.413343 IP 192.168.56.101 > 192.168.56.102: ICMP echo reply, id 7182, seq 0, length 8

```

We can see that the target has responded with an ICMP echo reply packet.

- Besides giving the options in the command line, you can also use `hping3` interactively. Open the console and type `hping3`. You will then see a prompt where you can type your Tcl commands.

Note

The following are several resources for Tcl:

<http://www.invece.org/tclwise/>

<http://wiki.tcl.tk/>

For the preceding example, the following is the corresponding Tcl script:

```
hping send {ip(daddr=192.168.56.101)+icmp(type=8,code=0)}
```

Open a command-line window and give the following command to get a response from the target server:

```
hping recv eth0
```

After that, open another command-line window to input the sending request.

The following screenshot shows the response received:

```

root@kali:~# hping3
hping3> hping recv eth0
ip(ihl=0x0,ver=0x0,tos=0x00,totlen=0,id=0,fragoff=0,mf=0,df=0,rf=0,ttl=0,proto=0,cksum=0x0000,saddr=0.0.0.0,daddr=0.0.0.0)

```

- You can also use `hping3` to check for a firewall rule. Let's suppose you have the following firewall rules:
 - Accept any TCP packets directed to port 22 (SSH)
 - Accept any TCP packets related with an established connection
 - Drop any other packets

To check these rules, you can give the following command in `hping3` in order to send an ICMP echo request packet:

```
hping3 -1 192.168.56.101 -c 1
```

The following code is the result:

```

HPING 192.168.56.101 (eth0 192.168.56.101): icmp mode set, 28 headers
+ 0 data bytes
--- 192.168.56.101 hping statistic ---
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms

```

We can see that the target machine has not responded to our ping probe.

Send a TCP packet with the SYN flag set to port **22** , and we will get a result as shown in the following screenshot:

```
root@kali:~# hping3 192.168.56.101 -c 1 -S -p 22 -s 6060
HPING 192.168.56.101 (eth0 192.168.56.101): S set, 40 headers + 0 data bytes
len=46 ip=192.168.56.101 ttl=64 DF id=0 sport=22 flags=SA seq=0 win=5840 rtt=2.5 ms

--- 192.168.56.101 hping statistic ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 2.5/2.5/2.5 ms
```

From the preceding screenshot, we can see that the target machine's firewall allows our syn packet to reach port **22** .

Let's check whether the UDP packet is allowed to reach port **22** :

```
root@kali:~# hping3 -2 192.168.56.101 -c 1 -S -p 22 -s 6060
HPING 192.168.56.101 (eth0 192.168.56.101): udp mode set, 28 headers + 0 data bytes

--- 192.168.56.101 hping statistic ---
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

From the preceding screenshot, we can see that the target machine's firewall does not allow our UDP packet to reach port **22** . There are other things that you can do with **hping3** , but in this chapter, we'll only discuss a small subset of its capabilities. If you want to learn more, you can consult the **hping3** documentation site at <http://wiki.hping.org>.

nping

The **nping** tool is a tool that allows users to generate network packets of a wide range of protocols (TCP, UDP, ICMP, and ARP). You can also customize the fields in the protocol headers, such as the source and destination port for TCP and UDP. The difference between **nping** and other similar tools such as **ping** is that **nping** supports multiple target hosts and port specification.

Besides, it can be used to send an ICMP echo request just like in the **ping** command; **nping** can also be used for network stress testing, **Address Resolution Protocol (ARP)** poisoning, and the denial of service attacks.

In Kali Linux, **nping** is included with the Nmap package.

The following are several probe modes supported by **nping** :

No.	Mode	Description
1	--tcp-connect	This is an unprivileged TCP connect
2	--tcp	This is a TCP mode
3	--udp	This is a UDP mode
4	--icmp	This is an ICMP mode (default)
5	--arp	This is an ARP/RARP mode
6	--tr	This is a traceroute mode (it can only be used in the TCP/UDP/ICMP mode)

At the time of this writing, there is no Kali Linux menu yet for **nping** . So, you need to open a console and type **nping** . This will display the usage and options' description.

In order to use **nping** to send an ICMP echo request to the target machines **192.168.56.100** , **192.168.56.101** , and

192.168.56.102 , you can give the following command:

```
nping -c 1 192.168.56.100-102
```

The following screenshot shows the command output:

```
Starting Nping 0.6.25 ( http://nmap.org/nping ) at 2013-06-28 20:48 WIT
SENT (0.0087s) ICMP 192.168.56.101 > 192.168.56.100 Echo request (type=8/code=0) ttl=64 id=32821 iplen=28
SENT (1.0109s) ICMP 192.168.56.101 > 192.168.56.101 Echo request (type=8/code=0) ttl=64 id=32821 iplen=28
SENT (2.0134s) ICMP 192.168.56.101 > 192.168.56.102 Echo request (type=8/code=0) ttl=64 id=32821 iplen=28
RCVD (2.0153s) ICMP 192.168.56.102 > 192.168.56.101 Echo reply (type=0/code=0) ttl=64 id=62113 iplen=28

Statistics for host 192.168.56.100:
| Probes Sent: 1 | Rcvd: 0 | Lost: 1 (100.00%)
|_ Max rtt: N/A | Min rtt: N/A | Avg rtt: N/A
Statistics for host 192.168.56.101:
| Probes Sent: 1 | Rcvd: 0 | Lost: 1 (100.00%)
|_ Max rtt: N/A | Min rtt: N/A | Avg rtt: N/A
Statistics for host 192.168.56.102:
| Probes Sent: 1 | Rcvd: 1 | Lost: 0 (0.00%)
|_ Max rtt: 1.461ms | Min rtt: 1.461ms | Avg rtt: 1.461ms
Raw packets sent: 3 (84B) | Rcvd: 1 (46B) | Lost: 2 (66.67%)
Tx time: 2.00769s | Tx bytes/s: 41.84 | Tx pkts/s: 1.49
Rx time: 2.00856s | Rx bytes/s: 22.90 | Rx pkts/s: 0.50
Nping done: 3 IP addresses pinged in 2.02 seconds
```

From the preceding screenshot, we know that only the 192.168.56.102 machine is sending back the ICMP echo reply packet.

If the machine is not responding to the ICMP echo request packet as shown in the following output, you can still find out whether it is alive by sending a TCP SYN packet to an open port in that machine:

```
root@kali:~# nping -c 1 192.168.56.102

Starting Nping 0.6.40 ( http://nmap.org/nping ) at 2013-11-08 12:36 WIT
SENT (0.0036s) ICMP [192.168.56.101 > 192.168.56.102 Echo request (type=8/code=0) id=40235 seq=1] IP [ttl=64 id=59056 iplen=28 ]

Max rtt: N/A | Min rtt: N/A | Avg rtt: N/A
Raw packets sent: 1 (28B) | Rcvd: 0 (0B) | Lost: 1 (100.00%)
Nping done: 1 IP address pinged in 1.01 seconds
```

For example, to send one (-c 1) TCP packet (--tcp) to the IP address 192.168.56.102 port 22 (-p 22), you can give the following command:

```
nping --tcp -c 1 -p 22 192.168.56.102
```

Of course, you need to guess the ports which are open. We suggest that you try with the common ports, such as 21 , 22 , 23 , 25 , 80 , 443 , 8080 , and 8443 .

The following screenshot shows the result of the mentioned example:

```
root@kali:~# nping --tcp -c 1 -p 22 192.168.56.102

Starting Nping 0.6.40 ( http://nmap.org/nping ) at 2013-11-08 12:38 WIT
SENT (0.0030s) TCP 192.168.56.101:10561 > 192.168.56.102:22 S ttl=64 id=18944 ip len=40 seq=1823950621 win=1480
RCVD (0.0043s) TCP 192.168.56.102:22 > 192.168.56.101:10561 SA ttl=64 id=0 iplen=44 seq=793586661 win=5840 <mss 1460>

Max rtt: 1.122ms | Min rtt: 1.122ms | Avg rtt: 1.122ms
Raw packets sent: 1 (40B) | Rcvd: 1 (46B) | Lost: 0 (0.00%)
Nping done: 1 IP address pinged in 1.00 seconds
```

From the preceding result, we can see that the remote machine (192.168.56.102) is alive because when we sent the TCP packet to port 22 , the target machine responded.

alive6

If you want to discover which machines are alive in an IPv6 environment, you can't just ask the tool to scan the whole network. This is because the address space is very huge. You may find that the machines have a 64-bit network range. Trying to discover the machines sequentially in this network will require at least 2^{64} packets. Of course, this is not a feasible task in the real world.

Fortunately, there is a protocol called ICMPv6 Neighbor Discovery. This protocol allows an IPv6 host to discover the link-local and autoconfigured addresses of all other IPv6 systems on the local network. In short, you can use this protocol to find a live host on the local network subnet.

To help you do this, there is a tool called `alive6`, which can send an ICMPv6 probe and is able to listen to the responses. This tool is part of the THC-IPv6 Attack Toolkit developed by van Hauser from The Hackers Choice (<http://freeworld.thc.org/thc-ipv6/>) group.

To access `alive6`, go to the console and type `alive6`. This will display the usage information.

Suppose you want to find the active IPv6 systems on your local IPv6 network, the following command can be given with the assumption that the `eth0` interface is connected to the LAN:

```
alive6 -p eth0
```

The following command lines are the result:

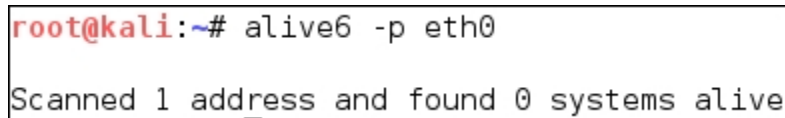
```
Alive: fe80::a00:27ff:fe43:1518 [ICMP echo-reply]
```

```
Scanned 1 address and found 1 system alive
```

To mitigate against this, you can block the ICMPv6 echo request with the following `ip6tables` command:

```
ip6tables -A INPUT -p ipv6-icmp --type icmpv6-type 128 -j DROP
```

The following screenshot is the result after the target machine configures the `ip6tables` rule:



```
root@kali:~# alive6 -p eth0
Scanned 1 address and found 0 systems alive
```

detect-new-ip6

This tool can be used if you want to detect the new IPv6 address joining a local network. This tool is part of the THC-IPv6 Attack Toolkit developed by van Hauser from The Hackers Choice group.

To access `detect-new-ipv6`, go to the console and type `detect-new-ipv6`. This will display the usage information.

Following is a simple usage of this tool; we want to find the new IPv6 address that joined the local network:

```
detect-new-ip6 eth0
```

The following is the result of that command:

```
Started ICMP6 DAD detection (Press Control-C to end) ...
Detected new ip6 address: fe80::a00:27ff:fe43:1518
```

passive_discovery6

This tool can be used if you want to sniff out the local network to look for the IPv6 address. This tool is part of the THC-IPv6 Attack Toolkit developed by van Hauser from The Hackers Choice group. Getting the IPv6 address without being detected by an IDS can be useful.

To access `passive_discovery6`, go to the console and type `passive_discovery6`. This will display the usage information on the screen.

The following command is an example of running this tool:

```
passive_discovery6 eth0
```

The following screenshot is the result of that command:

```

root@kali:~# passive_discovery6 eth0
Started IPv6 passive system detection (Press Control-C to end) ...
Detected: fe80::31ad:1227:d1d3:a002
Detected: fe80::a00:27ff:fe43:1518

```

This tool simply waits for the ARP request/reply by monitoring the network, and then it maps the answering hosts. The following are the IPv6 addresses that can be discovered by this tool on the network:

- `fe80::31ad:1227:d1d3:a002`
- `fe80::a00:27ff:fe43:1518`

nbtscan

If you are doing an internal penetration testing on a Windows environment, the first thing you want to do is get the NetBIOS information. One of the tools that can be used to do this is `nbtscan`.

The `nbtscan` tool will produce a report that contains the IP address, NetBIOS computer name, services available, logged in username, and MAC address of the corresponding machines. The NetBIOS name is useful if you want to access the service provided by the machine using the NetBIOS protocol that is connected to an open share. Be careful as using this tool will generate a lot of traffic and it may be logged by the target machines.

Note

To find the meaning of each service in the NetBIOS report, you may want to consult the Microsoft Knowledge Based on the *NetBIOS Suffixes (16th Character of the NetBIOS Name)* article at <http://support.microsoft.com/kb/163409>.

To access `nbtscan`, you can open the console and type `nbtscan`.

As an example, I want to find out the NetBIOS name of the computers located in my network (`192.168.1.0/24`). The following is the command to be used:

```
nbtscan 192.168.1.1-254
```

The following is the result of that command:

```

Doing NBT name scan for addresses from 192.168.1.1-254
IP address      NetBIOS Name    Server    User          MAC address
-----
---
192.168.1.81    PC-001          <server>  <unknown>     00:25:9c:9f:b0:96
192.168.1.90    PC-003          <server>  <unknown>     00:00:00:00:00:00
...

```

From the preceding result, we are able to find three NetBIOS names, `PC-001`, `PC-003`, and `SRV-001`.

Let's find the service provided by these machines by giving the following command:

```
nbtscan -hv 192.168.1.1-254
```

Option `-h` will print the service in a human-readable name. While, option `-v` will give more verbose output information.

The following is the result of this command:

```

NetBIOS Name Table for Host 192.168.1.81:
PC-001      Workstation Service
PC-001      File Server Service
WORKGROUP   Domain Name
WORKGROUP   Browser Service Elections
Adapter address: 00:25:9c:9f:b0:96

```

NetBIOS Name Table for Host 192.168.1.90:

```

PC-003      Workstation Service
PC-003      Messenger Service
PC-003      File Server Service
__MSBROWSE__ Master Browser
WORKGROUP   Domain Name
WORKGROUP   Browser Service Elections
WORKGROUP   Domain Name
WORKGROUP   Master Browser

```

Adapter address: 00:00:00:00:00:00

...

From the preceding result, we can see that there are two services available on **PC-001** : **Workstation** and **File Server** . While in **PC-003** , there are three services available: **Workstation** , **Messenger** , and **File Server** . In our experience, this information is very useful because we know which machine has a file sharing service. Next, we can continue to check whether the file sharing services are open so that we can access the files stored on those file sharing services.

Username: Palm Beach State College IP Holder **Book:** Kali Linux – Assuring Security by Penetration Testing. No part of any chapter or book may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher of the book or chapter. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates these Terms of Service is strictly prohibited. Violators will be prosecuted to the full extent of U.S. Federal and Massachusetts laws.

OS fingerprinting

After we know that the target machine is a live, we can then find out the operating system used by the target machine. This method is commonly known as **Operating System (OS) fingerprinting**. There are two methods of doing OS fingerprinting: **active** and **passive**.

In the active method, the tool sends network packets to the target machine and then determines the operating system of the target machine based on the analysis done on the response it has received. The advantage of this method is that the fingerprinting process is fast. However, the disadvantage is that the target machine may notice our attempt to get its operating system's information.

To overcome the active method's disadvantage, there exists a passive method of OS fingerprinting. This method was pioneered by Michal Zalewski when he released a tool called **p0f**. The disadvantage of the passive method is that the process will be slower than the active method.

In this section, we will describe a couple of tools that can be used for OS fingerprinting.

p0f

The **p0f** tool is used to fingerprint an operating system passively. It can be used to identify an operating system on the following machines:

- Machines that connect to your box (SYN mode; this is the default mode)
- Machines you connect to (SYN+ACK mode)
- Machines you cannot connect to (RST+ mode)
- Machines whose communications you can observe

The **p0f** tool works by analyzing the TCP packets sent during the network activities. Then, it gathers the statistics of special packets that are not standardized by default by any corporations. An example is that the Linux kernel uses a 64-byte ping datagram, whereas the Windows operating system uses a 32-byte ping datagram; or the **Time To Live (TTL)** value. For Windows, the TTL value is 128, while for Linux this TTL value varies between the Linux distributions. These information are then used by **p0f** to determine the remote machine's operating system.

Note

When using the **p0f** tool included with Kali Linux, we were not able to fingerprint the operating system on a remote machine. We figured out that the **p0f** tool has not updated its fingerprint database. Unfortunately, we couldn't find the latest version of the fingerprint database. So, we used **p0f** v3 (Version 3.06b) instead. To use this version of **p0f**, just download the TARBALL file from <http://lcamtuf.coredump.cx/p0f3/releases/p0f-3.06b.tgz> and compile the code by running the **build.sh** script. By default, the fingerprint database file (**p0f.fp**) location is in the current directory. If you want to change the location, for example, if you want to change the location to **/etc/p0f/p0f.fp**, you need to change this in the **config.h** file and recompile **p0f**. If you don't change the location, you may need to use the **-f** option to define the fingerprint database file location.

To access **p0f**, open a console and type **p0f -h**. This will display its usage and options' description.

Let's use **p0f** to identify the operating system used in a remote machine we are connecting to. Just type the following command in your console:

```
p0f -f /etc/p0f/p0f.fp -o p0f.log
```

This will read the fingerprint database from the **/etc/p0f/p0f.fp** file and save the log information to the **p0f.log** file. It will then display the following information:

```
--- p0f 3.06b by Michal Zalewski <lcamtuf@coredump.cx> ---

[+] Closed 1 file descriptor.
[+] Loaded 314 signatures from '/etc/p0f/p0f.fp'.
[+] Intercepting traffic on default interface 'eth0'.
[+] Default packet filtering configured [+VLAN].
[+] Log file 'p0f.log' opened for writing.
```

[+] Entered main event loop.

Next, you need to generate network activities involving a TCP connection, such as browsing to the remote machine or letting the remote machine to connect to your machine.

If **p0f** has successfully fingerprinted the operating system, you will see information of the remote machine's operating system in the console and in the logfile (**p0f.log**).

Following is the information displayed to the console:

```

.-[ 192.168.56.101/42819 -> 192.168.56.102/80 (syn) ]-
|
| client    = 192.168.56.101/42819
| os        = Linux 3.x
| dist      = 0
| params    = none
| raw_sig   = 4:64+0:0:1460:mss*10,7:mss,sok,ts,nop,ws:df,id+:0
|
\-----

.-[ 192.168.56.101/42819 -> 192.168.56.102/80 (mtu) ]-
|
| client    = 192.168.56.101/42819
| link      = Ethernet or modem
| raw_mtu   = 1500
|
\-----

.-[ 192.168.56.101/42819 -> 192.168.56.102/80 (syn+ack) ]-
|
| server    = 192.168.56.102/80
| os        = Linux 2.6.x
| dist      = 0
| params    = none
| raw_sig   = 4:64+0:0:1460:mss*4,5:mss,sok,ts,nop,ws:df:0
|
\-----

.-[ 192.168.56.101/42819 -> 192.168.56.102/80 (mtu) ]-
|
| server    = 192.168.56.102/80
| link      = Ethernet or modem
| raw_mtu   = 1500
|
\-----

.-[ 192.168.56.101/42819 -> 192.168.56.102/80 (http request) ]-
|
| client    = 192.168.56.101/42819
| app       = Firefox 10.x or newer
| lang      = English
| params    = none
| raw_sig   = 1:Host,User-Agent,Accept=[text/html,application

```

```

//xhtml+xml,application/xml;q=0.9,*/*;q=0.8],Accept-Language=[en-US,en;q=0.5],Accept-Encoding=[gzip, deflate],Connection=[keep-alive]:Accept-Charset,Keep-Alive:Mozilla/5.0 (X11; Linux x86_64; rv:18.0) Gecko/20100101 Firefox/18.0 Iceweasel/18.0.1
|
`-----

```

```

.-[ 192.168.56.101/42819 -> 192.168.56.102/80 (http response) ]-
|
| server    = 192.168.56.102/80
| app       = Apache 2.x
| lang      = none
| params    = none
| raw_sig   = 1:Date,Server,X-Powered-By=[PHP/5.2.4-2ubuntu5.10],?Content-Length,Keep-Alive=[timeout=15, max=100],Connection=[Keep-Alive],Content-Type:Accept-Ranges:Apache/2.2.8 (Ubuntu) DAV/2
|
`-----

```

The following screenshot shows the content of the logfile:

```

[2013/06/28 22:47:57] mod=syn|cli=192.168.56.101/42819|srv=192.168.56.102/80|subj=cli|os=Linux 3.x|dist=0|params=none|raw_sig=4:64+0:0:1460:mss*10,7:mss,sok,ts,nop,ws:df,id+:0
[2013/06/28 22:47:57] mod=mtu|cli=192.168.56.101/42819|srv=192.168.56.102/80|subj=cli|link=Ethernet or modem|raw_mtu=1500
[2013/06/28 22:47:57] mod=syn+ack|cli=192.168.56.101/42819|srv=192.168.56.102/80|subj=srv|os=Linux 2.6.x|dist=0|params=none|raw_sig=4:64+0:0:1460:mss*4,5:mss,sok,ts,nop,ws:df:0
[2013/06/28 22:47:57] mod=mtu|cli=192.168.56.101/42819|srv=192.168.56.102/80|subj=srv|link=Ethernet or modem|raw_mtu=1500
[2013/06/28 22:47:57] mod=http request|cli=192.168.56.101/42819|srv=192.168.56.102/80|subj=cli|app=Firefox 10.x or newer|lang=English|params=none|raw_sig=1:Host,User-Agent,Accept=[text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8],Accept-Language=[en-US,en;q=0.5],Accept-Encoding=[gzip, deflate],Connection=[keep-alive]:Accept-Charset,Keep-Alive:Mozilla/5.0 (X11; Linux x86_64; rv:18.0) Gecko/20100101 Firefox/18.0 Iceweasel/18.0.1
[2013/06/28 22:47:57] mod=http response|cli=192.168.56.101/42819|srv=192.168.56.102/80|subj=srv|app=Apache 2.x|lang=none|params=none|raw_sig=1:Date,Server,X-Powered-By=[PHP/5.2.4-2ubuntu5.10],?Content-Length,Keep-Alive=[timeout=15, max=100],Connection=[Keep-Alive],Content-Type:Accept-Ranges:Apache/2.2.8 (Ubuntu) DAV/2

```

Based on the preceding result, we know that the target is a **Linux 2.6** machine.

The following screenshot shows the information from the target machine:

```

root@metasploitable:/home/msfadmin# uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux

```

By comparing this information, we know that **p0f** got the OS information correctly. The remote machine is using Linux Version 2.6.

You can stop **p0f** by pressing the **Ctrl + C** key combination.

Nmap

Nmap is a very popular and capable port scanner. Besides this, it can also be used to fingerprint a remote machine's operating system. It is an active fingerprinting tool. To use this feature, you can give the **-O** option to the **nmap** command.

For example, if we want to fingerprint the operating system used on the **192.168.56.102** machine, we use the following command:

```
nmap -O 192.168.56.102
```

The following screenshot shows the result of this command:


```
MAC Address: 08:00:27:43:15:18 (Cadmus Computer Systems)
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 http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 15.35 seconds
```

Nmap was able to get the correct operating system information after fingerprinting the operating system of a remote machine.

We will talk more about Nmap in a later chapter.

Username: Palm Beach State College IP Holder **Book:** Kali Linux – Assuring Security by Penetration Testing. No part of any chapter or book may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher of the book or chapter. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates these Terms of Service is strictly prohibited. Violators will be prosecuted to the full extent of U.S. Federal and Massachusetts laws.

Summary

In this chapter, we discussed the target discovery process. We started by discussing the purpose of target discovery: identifying the target machine and finding out the operating system used by the target machine. Then, we continued with the tools included with Kali Linux that can be used for identifying target machines.

We discussed the following tools: `ping` , `arping` , `fping` , `hping3` , `nping` , and `nbtscan` . We also discussed several tools specially developed to be used in an IPv6 environment, such as `alive6` , `detect-new-ip6` , and `passive_discovery6` .

At the end of this chapter, you learned about the tools that can be used to do OS fingerprinting: `p0f` , and briefly about the `nmap` capabilities for doing active operating system fingerprinting.

In the next chapter, we will talk about target enumeration and describe the tools included in Kali Linux that can be used for this purpose.