Host Discovery with nmap

By: Mark Wolfgang
moonpie@moonpie.org

November 2002

Table of Contents

1. Introduction

As a Computer Security Engineer that regularly conducts external penetration tests, a recurring challenge seems to arise when assessing organizations with a large allocation of IP address space. What does one do when faced with multiple class B's, a few class C's, and a limited amount of time? Do you stick all of the address space in your favorite scanner and hit the *Go* button, wait till it's done and hope the results are accurate? How can you be sure that your scanner found all the hosts that are accessible? Do you even know the method your scanner uses to discover which hosts are alive?

This document attempts to answer the above questions, and will illustrate (at a very technical level) the methodology that I use to accurately discover which hosts are accessible prior to conducting port scanning or a vulnerability assessment.

Note: Some may say that unless one performs a scan on all 65535 TCP and UDP ports on every possible IP address in the range, that the penetration tester isn't being thorough enough. While I do agree that in order to be completely thorough, one must perform a scan as stated, but I have rarely, if ever had the luxury of performing such a scan, as it usually takes a considerable amount of time. An underlying theme about Information Security is about striking a balance and weighing the pros and cons. If being absolutely thorough and time is of no consideration, you'll more than likely want to run a full, blind scan on all IP addresses. If however, a balance can be struck between being thorough and completing the project on time, read on – you may learn some techniques to improve both the accuracy and efficiency of your scans.

1.1 What is Host Discovery?

Host discovery is a term I'll use to describe a certain phase of a penetration test, where one attempts to determine the accessible hosts on a network. Many times if a firewall ruleset is written explicitly, it is difficult to accurately determine the number of hosts that are behind a firewall. A colleague of mine recently ran a high-priced commercial scanner against a class C and found only one host. Using the techniques outlined in this paper, I was able to determine that there was not just one, but seventeen hosts in this particular DMZ. This commercial scanner has very few options for host discovery, and is not very configurable when it comes to fine tuning the discovery method.

Since this paper is about nmap and host discovery, we'll talk specifically about how nmap does its discovery, and we'll learn how to use nmap's options to improve the discovery phase of a penetration test or vulnerability assessment.

To start off, let's dissect the following very basic nmap command:

```
nmap –sS –O 172.26.1.0/29
```

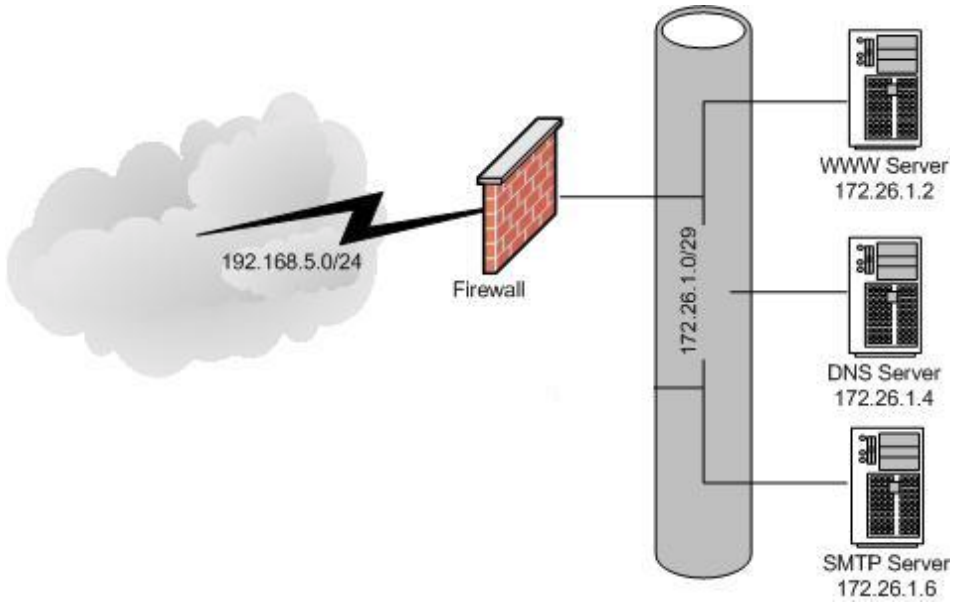
There are three distinct phases with the above nmap command. They are:

1) Host discovery
2) Port scanning
3) OS fingerprinting

Since this paper is focused on host discovery, we will take an in-depth look at the first phase of the above nmap command, skipping the latter two.

Note: It is possible to disable the discovery phase of the scan (with the –P0 option), and tell nmap to move directly on to the port scan phase.

In this paper we will use a DMZ environment with a variety of different firewall rulesets to illustrate the best methods for discovering hosts behind a firewall. The DMZ architecture we will use throughout this paper is depicted in the following image.



Here we have a typical DMZ with a firewall filtering inbound traffic. In our scenarios we will use "pseudo-rulesets" to keep the rules readable. The actual syntax from the rules are a mix between PF and engrish, so don't get hung up on the accuracy of them – they just need to be readable. Also, the version of nmap I used for this testing was 3.00

Our scanning host sits on the 192.168.5.0/24 network and has the IP address of 192.168.5.20.

Unless otherwise stated, we will use the following nmap command for all discovery scans:

nmap –sP 172.26.1.0/29

The –sP option specifies that only a discovery will performed, and is the same discovery method used in a default nmap scan.

2. Exploring nmap's Default Behavior

Before we learn how and why it may be necessary to modify nmap's behavior, we should have a solid understanding of the default behavior, and how it may be insufficient in performing host discovery.

When a port scan (nmap –sS target) or a "ping sweep" (nmap –sP target) is run against a target network or host, nmap simultaneously sends out ICMP echo request packets and "TCP pings" to all targets within the scope of the scan. The term "TCP ping" can be described as a TCP packet with the ACK flag set, destined for port 80 of the target host(s). The desired response from an accessible host is either a TCP packet with the RST flag set, or an ICMP echo reply, indicating that the host is alive. No response would indicate that the host is not alive, or is being protected by a firewall, and is therefore unreachable on port 80.

Only when nmap has determined that the host is in fact reachable does it attempt to portscan the target. The default nmap discovery method works well in certain circumstances, but should not be completely relied upon to determine the accessible hosts. Fortunately, nmap is very flexible and allows us to customize just about every aspect of the discovery.

The following depicts a default nmap discovery of one host.

Our command:

Nmap –sP 172.26.1.1

tcpdump Output:

```
09:26:49.324016 192.168.5.20 > 172.26.1.1: ICMP: echo request
09:26:49.324083 192.168.5.20.40435 > 172.26.1.1.http: . ack 1942297083 win 3072
```

Above we see the ICMP ping and the TCP ping being sent.

2.1 Scenario 1 – Firewall With No Filtering

Our first scenario will demonstrate how the discovery process works with no filtering being performed by the firewall. (i.e. the firewall is simply acting as a router).

Our Command:

nmap –sP 172.26.1.0/29

Firewall Ruleset:

pass from any to any

tcpdump Output:

```
08:59:58.840249 192.168.5.20 > 172.26.1.0: ICMP: echo request
08:59:58.840667 192.168.5.20.60923 > 172.26.1.0.http: . ack 2990889584 win 3072
08:59:58.840726 192.168.5.20 > 172.26.1.1: ICMP: echo request
08:59:58.840764 192.168.5.20.60923 > 172.26.1.1.http: . ack 1015938099 win 3072
08:59:58.840801 192.168.5.20 > 172.26.1.2: ICMP: echo request
08:59:58.840838 192.168.5.20.60923 > 172.26.1.2.http: . ack 1228729075 win 3072
08:59:58.840876 192.168.5.20 > 172.26.1.3: ICMP: echo request
08:59:58.840914 192.168.5.20.60923 > 172.26.1.3.http: . ack 1769982015 win 3072
08:59:58.840952 192.168.5.20 > 172.26.1.4: ICMP: echo request
08:59:58.840989 192.168.5.20.60923 > 172.26.1.4.http: . ack 1859940754 win 3072
08:59:58.841027 192.168.5.20 > 172.26.1.5: ICMP: echo request
08:59:58.841064 192.168.5.20.60923 > 172.26.1.5.http: . ack 1596045207 win 3072
08:59:58.841103 192.168.5.20 > 172.26.1.6: ICMP: echo request
08:59:58.841140 192.168.5.20.60923 > 172.26.1.6.http: . ack 550856434 win 3072
08:59:58.841178 192.168.5.20 > 172.26.1.7: ICMP: echo request
08:59:58.841215 192.168.5.20.60923 > 172.26.1.7.http: . ack 2476756145 win 3072
08:59:58.841886 172.26.1.2 > 192.168.5.20: ICMP: echo reply
08:59:58.842149 172.26.1.4 > 192.168.5.20: ICMP: echo reply
08:59:58.842377 172.26.1.2.http > 192.168.5.20.60923: R
1228729075:1228729075(0) win 0 (DF)
08:59:58.842699 192.168.5.5 > 192.168.5.20: ICMP: echo reply
08:59:58.842905 172.26.1.4.http > 192.168.5.20.60923: R
1859940754:1859940754(0) win 0 (DF)
08:59:58.843263 172.26.1.6 > 192.168.5.20: ICMP: echo reply
08:59:58.843487 172.26.1.6.http > 192.168.5.20.60923: R 550856434:550856434(0)
win 0 (DF)
```

Results:

All hosts found.

In the above tcpdump output we can see exactly how nmap does its job. It sends out the ICMP and TCP packets to all hosts within scope (172.26.1.0/29) and waits for replies. The replies from the accessible hosts are highlighted in yellow.

2.2 Scenario 2 – Firewall With a Generic Ruleset

This scenario illustrates how the default discovery method works with a rather generic ruleset.

Our Command:

nmap –sP 172.26.1.0/29

Firewall Ruleset:

```
pass from any to any proto tcp port 80
pass from any to any proto tcp port 53
pass from any to any proto tcp port 25
drop all
```

tcpdump Output:

```
09:12:21.505016 192.168.5.20 > 172.26.1.0: ICMP: echo request
09:12:21.505125 192.168.5.20.60212 > 172.26.1.0.http: . ack 3755150488 win 3072
09:12:21.505166 192.168.5.20 > 172.26.1.1: ICMP: echo request
09:12:21.505204 192.168.5.20.60212 > 172.26.1.1.http: . ack 4073218537 win 3072
09:12:21.505242 192.168.5.20 > 172.26.1.2: ICMP: echo request
09:12:21.505280 192.168.5.20.60212 > 172.26.1.2.http: . ack 3464075465 win 3072
09:12:21.505318 192.168.5.20 > 172.26.1.3: ICMP: echo request
09:12:21.505355 192.168.5.20.60212 > 172.26.1.3.http: . ack 962650084 win 3072
09:12:21.505393 192.168.5.20 > 172.26.1.4: ICMP: echo request
09:12:21.505430 192.168.5.20.60212 > 172.26.1.4.http: . ack 337683576 win 3072
09:12:21.505468 192.168.5.20 > 172.26.1.5: ICMP: echo request
09:12:21.505505 192.168.5.20.60212 > 172.26.1.5.http: . ack 1839298263 win 3072
09:12:21.505544 192.168.5.20 > 172.26.1.6: ICMP: echo request
09:12:21.505581 192.168.5.20.60212 > 172.26.1.6.http: . ack 1701634905 win 3072
09:12:21.505619 192.168.5.20 > 172.26.1.7: ICMP: echo request
09:12:21.505656 192.168.5.20.60212 > 172.26.1.7.http: . ack 4287112447 win 3072
09:12:21.506577 172.26.1.2.http > 192.168.5.20.60212: R 3464075465:3464075465(0) win 0 (DF)
09:12:21.506830 172.26.1.6.http > 192.168.5.20.60212: R 1701634905:1701634905(0) win 0 (DF)
09:12:21.507104 172.26.1.4.http > 192.168.5.20.60212: R 337683576:337683576(0) win 0 (DF)
```

Results:

All hosts found.

This firewall ruleset is pretty poor, but not that uncommon. It is not stateful, so our TCP pings get through without a problem. Our ICMP packets however, do not get through due to the “cleanup” rule at the end.

2.3 Scenario 3 – Firewall With Specific Rules

This scenario illustrates how the default discovery method works with a more specific ruleset.

Our Command:

nmap -sP 172.26.1.0/29

Firewall Ruleset:

pass from any to 172.26.1.2 proto tcp port 80
pass from any to 172.26.1.4 proto tcp port 53
pass from any to 172.26.1.6 proto tcp port 25
drop all

tcpdump Output:

```
08:05:07.733225 192.168.5.20 > 172.26.1.0: ICMP: echo request
08:05:07.733334 192.168.5.20.44273 > 172.26.1.0.http: . ack 1621467562 win 2048
08:05:07.733375 192.168.5.20 > 172.26.1.1: ICMP: echo request
08:05:07.733412 192.168.5.20.44273 > 172.26.1.1.http: . ack 1213996683 win 2048
08:05:07.733450 192.168.5.20 > 172.26.1.2: ICMP: echo request
08:05:07.733487 192.168.5.20.44273 > 172.26.1.2.http: . ack 3921129299 win 2048
08:05:07.733525 192.168.5.20 > 172.26.1.3: ICMP: echo request
08:05:07.733561 192.168.5.20.44273 > 172.26.1.3.http: . ack 193217699 win 2048
08:05:07.733598 192.168.5.20 > 172.26.1.4: ICMP: echo request
08:05:07.733635 192.168.5.20.44273 > 172.26.1.4.http: . ack 3130918107 win 2048
08:05:07.733672 192.168.5.20 > 172.26.1.5: ICMP: echo request
08:05:07.733709 192.168.5.20.44273 > 172.26.1.5.http: . ack 638273071 win 2048
08:05:07.733746 192.168.5.20 > 172.26.1.6: ICMP: echo request
08:05:07.733783 192.168.5.20.44273 > 172.26.1.6.http: . ack 4151673259 win 2048
08:05:07.733820 192.168.5.20 > 172.26.1.7: ICMP: echo request
08:05:07.733856 192.168.5.20.44273 > 172.26.1.7.http: . ack 228721671 win 2048
08:05:07.734611 172.26.1.2.http > 192.168.5.20.44273: R 3921129299:3921129299(0) win 0 (DF)
```

Results:

1 host found.

Since our filtering is now specific, the default nmap discovery method is not sufficient. The only packet that gets through the firewall is the TCP ping destined for the WWW server.

2.4 Scenario 4 – Stateful Firewall with Specific Rules

In this scenario our firewall performs stateful inspection, and our firewall ruleset is very specific.

Our Command:

nmap -sP 172.26.1.0/29

Firewall Ruleset:

```
pass from any to 172.26.1.2 proto tcp port 80 keep state
pass from any to 172.26.1.4 proto tcp port 53 keep state
pass from any to 172.26.1.6 proto tcp port 25 keep state
drop all
```

tcpdump Output:

```
08:46:23.548456 192.168.5.20.44390 > 172.26.1.2.http: . ack 3476163011 win 2048
08:46:23.548468 192.168.5.20 > 172.26.1.3: ICMP: echo request
08:46:23.548501 192.168.5.20.44390 > 172.26.1.3.http: . ack 1149703540 win 2048
08:46:23.548559 192.168.5.20 > 172.26.1.4: ICMP: echo request
08:46:23.548596 192.168.5.20.44390 > 172.26.1.4.http: . ack 1314586500 win 2048
08:46:23.548635 192.168.5.20 > 172.26.1.5: ICMP: echo request
08:46:23.548673 192.168.5.20.44390 > 172.26.1.5.http: . ack 2068473993 win 2048
08:46:23.548712 192.168.5.20 > 172.26.1.6: ICMP: echo request
08:46:23.548749 192.168.5.20.44390 > 172.26.1.6.http: . ack 2732407633 win 2048
08:46:23.548789 192.168.5.20 > 172.26.1.7: ICMP: echo request
08:46:23.548825 192.168.5.20.44390 > 172.26.1.7.http: . ack 2518875875 win 2048
```

Results:

No hosts found.

Above we see the standard TCP pings and ICMP packets are sent, but this time we receive no reply from the hosts, as the “cleanup” rule drops the ICMP and the firewall drops the TCP ACK packets (TCP pings) because they are not part of a previously established connection.

It should now be clear why nmap’s default discovery method is insufficient with a firewall such as this one.

3. Understanding nmap's Discovery Options

In the preceding section we can see that nmap's default discovery options are not sufficient in two of the four scenarios. In order to become confident in using nmap to discover hosts, we must learn to use some of its advanced options.

3.1 Customizing TCP Pings

The default TCP ping uses a TCP packet with the ACK flag set with a destination port of 80 (WWW). Upon receiving this packet, a stateful firewall will look in its state table to see if the packet is part of a previously established connection. Once it finds that this is a rogue TCP packet, it will promptly discard the packet, preventing us from discovering the host. Firewalls that do not perform stateful inspection but have a specific ruleset will only pass the TCP ping through to authorized hosts. It is therefore necessary to customize the TCP ping.

Nmap allows us to customize most options within the TCP ping. The first thing one might want to do is set the SYN flag instead of the ACK flag. This is easily achieved by using the –PS option. A complete command might look like:

```
nmap –sP –PS 172.26.1.2
```

tcpdump Output:

```
10:48:13.656653 192.168.5.20.50992 > 172.26.1.2.http: S 3312451587:3312451587(0) win 2048
```

The above command still uses the default port 80, but will be passed by the stateful firewall (unless another rule prohibits it) due to the SYN flag being set. By adding a destination port to the –PS option, nmap allows you to further customize the discovery. A fully customized nmap discovery may look like:

```
nmap –sP –PS25  172.26.1.2
```

tcpdump Output:

```
10:49:50.436438 192.168.5.20.63376 > 172.26.1.2.smtp: S 948961283:948961283(0) win 4096
```

This command sends the TCP ping (with the SYN flag set) to port 25 instead of the default of port 80.

Another option one may consider when customizing TCP Pings is setting a specific source port on your TCP packets. This may not always work, but is surely worth a try. This will only work with very poorly written firewall rules, but consider the following pseudo ruleset:

```
pass from any to 172.26.1.2 proto tcp port 80 keep state
pass from any to 172.26.1.4 proto tcp port 53 keep state
pass from any to 172.26.1.6 proto tcp port 25 keep state
pass from any port 53 to any keep state
drop all
```

Our Command:

nmap -sP 172.26.1.0/29 –g 53

tcpdump Output:

```
10:52:02.083065 192.168.5.20 > 172.26.1.0: ICMP: echo request
10:52:02.083260 192.168.5.20.domain > 172.26.1.0.http: . ack 2177885259 win 3072
10:52:02.083301 192.168.5.20 > 172.26.1.1: ICMP: echo request
10:52:02.083346 192.168.5.20.domain > 172.26.1.1.http: . ack 2684323392 win 3072
10:52:02.083384 192.168.5.20 > 172.26.1.2: ICMP: echo request
10:52:02.083421 192.168.5.20.domain > 172.26.1.2.http: . ack 1438652920 win 3072
10:52:02.083459 192.168.5.20 > 172.26.1.3: ICMP: echo request
10:52:02.083496 192.168.5.20.domain > 172.26.1.3.http: . ack 771338950 win 3072
10:52:02.083534 192.168.5.20 > 172.26.1.4: ICMP: echo request
10:52:02.083570 192.168.5.20.domain > 172.26.1.4.http: . ack 3541039396 win 3072
10:52:02.083608 192.168.5.20 > 172.26.1.5: ICMP: echo request
10:52:02.083645 192.168.5.20.domain > 172.26.1.5.http: . ack 2586779353 win 3072
10:52:02.083683 192.168.5.20 > 172.26.1.6: ICMP: echo request
10:52:02.083719 192.168.5.20.domain > 172.26.1.6.http: . ack 45434507 win 3072
10:52:02.083757 192.168.5.20 > 172.26.1.7: ICMP: echo request
10:52:02.083794 192.168.5.20.domain > 172.26.1.7.http: . ack 1886752887 win 3072
10:52:02.084616 172.26.1.2.http > 192.168.5.20.domain: R 1438652920:1438652920(0) win 0 (DF)
10:52:02.084845 172.26.1.4.http > 192.168.5.20.domain: R 3541039396:3541039396(0) win 0 (DF)
10:52:02.085219 172.26.1.6.http > 192.168.5.20.domain: R 45434507:45434507(0) win 0 (DF)
```

Here we have a stateful firewall with specific rules, and also what appears to be a very harmless rule that permits DNS traffic from anywhere to any host in the DMZ. As indicated in the above tcpdump output, this rule does permit DNS traffic but also permits TCP packets from other sources to probe and scan the DMZ. Logic errors and vague rules in firewall rulesets are not uncommon, and can prove to be very useful in host discovery.

3.2 Customizing ICMP Messages

Though many firewalls will discard ICMP echo request messages, they may permit other types of ICMP traffic to pass unhindered. In addition to ICMP echo request messages, recent versions of nmap allow two other types of ICMP messages to be sent.

Using the –PP option, nmap will send ICMP timestamp requests (type 13) and expects ICMP timestamp replies (type 14) in return. Of course if a type 14 ICMP packet is received then nmap assumes the host is alive.

Our Command:

nmap –sP -PP 172.26.1.4

tcpdump Output:

```
13:32:05.780376 192.168.5.20 > 172.26.1.4: icmp: time stamp query id 47345 seq 0 (DF)
13:32:05.781066 172.26.1.4 > 192.168.5.20: icmp: time stamp reply id 47345 seq 0 : org 0x0
recv 0x3c339bd xmit 0x3c339bd
```

The –PM option sends ICMP address mask (netmask) requests (type 17) and expects an ICMP address mask reply (type 18) in return. Once again, if a type 18 packet is received, the host is alive.

Our Command:

nmap –sP -PM 172.26.1.4

tcpdump Output:

```
13:37:11.452204 192.168.5.20 > 172.26.1.4: icmp: address mask request (DF)
```

Note: Even if the firewall does pass these custom ICMP packets, not every operating system will comply with the request and may silenty discard the packet. Typically only routers will comply with netmask requests.

3.3 Bringing It All Together

Now that we have an understanding of how a default nmap discovery works, and how to further customize and use other features in nmap, we'll wrap it up with a final scenario. We'll use scenario four from above as our "worse case" situation, and will go through methods for determining all the hosts on the DMZ.

Firewall Ruleset

```
pass from any to 172.26.1.2 proto tcp port 80 keep state
pass from any to 172.26.1.4 proto tcp port 53 keep state
pass from any to 172.26.1.6 proto tcp port 25 keep state
drop all
```

Our command:

nmap –sP –PS80 172.26.1.0/29

tcpdump Output:

```
11:03:11.198359 192.168.5.20.49989 > 172.26.1.0.http: S 3857711107:3857711107(0) win 1024
11:03:11.198465 192.168.5.20.49989 > 172.26.1.1.http: S 3508535339:3508535339(0) win 1024
11:03:11.198506 192.168.5.20.49989 > 172.26.1.2.http: S 1017118803:1017118803(0) win 1024
11:03:11.198544 192.168.5.20.49989 > 172.26.1.3.http: S 832045179:832045179(0) win 1024
11:03:11.198582 192.168.5.20.49989 > 172.26.1.4.http: S 2873622691:2873622691(0) win 1024
11:03:11.198620 192.168.5.20.49989 > 172.26.1.5.http: S 1101529291:1101529291(0) win 1024
11:03:11.198658 192.168.5.20.49989 > 172.26.1.6.http: S 99614963:99614963(0) win 1024
11:03:11.198696 192.168.5.20.49989 > 172.26.1.7.http: S 3741843739:3741843739(0) win 1024
11:03:11.199453 172.26.1.2.http > 192.168.5.20.49989: S 92167158:92167158(0) ack
1017118804 win 5840 <mss 1460> (DF)
```

As expected, only our web server (172.26.1.2) replies. Now we'll change the destination port on the TCP Ping to 25.

Our command:

nmap –sP –PS25 172.26.1.0/29

tcpdump Output:

```
11:05:06.000617 192.168.5.20.38849 > 172.26.1.0.smtp: S 3544186883:3544186883(0) win 2048
11:05:06.000720 192.168.5.20.38849 > 172.26.1.1.smtp: S 4056940587:4056940587(0) win 2048
11:05:06.000759 192.168.5.20.38849 > 172.26.1.2.smtp: S 3272605779:3272605779(0) win 2048
11:05:06.000797 192.168.5.20.38849 > 172.26.1.3.smtp: S 4168614011:4168614011(0) win 2048
11:05:06.000835 192.168.5.20.38849 > 172.26.1.4.smtp: S 586154147:586154147(0) win 2048
11:05:06.000872 192.168.5.20.38849 > 172.26.1.5.smtp: S 3571974347:3571974347(0) win 2048
11:05:06.000910 192.168.5.20.38849 > 172.26.1.6.smtp: S 667418867:667418867(0) win 2048
11:05:06.000948 192.168.5.20.38849 > 172.26.1.7.smtp: S 902824219:902824219(0) win 2048
11:05:06.001909 172.26.1.6.smtp > 192.168.5.20.38849: S 203047548:203047548(0) ack
667418868 win 5840 <mss 1460> (DF)
```

Now we see that SMTP server (172.26.1.6) replies. Again, let's change the destination TCP port to 53 to probe for our DNS server.

Our command:

nmap –sP –PS53 172.26.1.0/29

tcpdump Output:

```
11:06:52.601522 192.168.5.20.51592 > 172.26.1.0.domain: S 2862088195:2862088195(0) win 4096
11:06:52.601623 192.168.5.20.51592 > 172.26.1.1.domain: S 3921149995:3921149995(0) win 4096
11:06:52.601662 192.168.5.20.51592 > 172.26.1.2.domain: S 3150970963:3150970963(0) win 4096
11:06:52.601699 192.168.5.20.51592 > 172.26.1.3.domain: S 4262985851:4262985851(0) win 4096
11:06:52.601736 192.168.5.20.51592 > 172.26.1.4.domain: S 3247440035:3247440035(0) win 4096
11:06:52.601773 192.168.5.20.51592 > 172.26.1.5.domain: S 1634730187:1634730187(0) win 4096
11:06:52.601811 192.168.5.20.51592 > 172.26.1.6.domain: S 947388659:947388659(0) win 4096
11:06:52.601848 192.168.5.20.51592 > 172.26.1.7.domain: S 2042102043:2042102043(0) win 4096
11:06:52.602957 172.26.1.4.domain > 192.168.5.20.51592: S 328239288:328239288(0) ack
3247440036 win 5840 <mss 1460> (DF)
```

Our DNS server (172.26.1.4) has replied, so now all of our hosts are accounted for. But wait a minute. Since we know all of the hosts and their services on our DMZ, we were able to know which ports we should send our TCP pings. When conducting an external assessment, it is highly unlikely that the client will tell you the systems and services available, making it difficult, but not impossible to discover the hosts.

What one must do to perform a complete discovery against a firewall such as the one above is to use TCP Ping sweeps with a variety of different destination and source ports. Destination ports of the most common Internet services would be a good start. Some good source ports to use would be 20, and 53. A script can be written to automate these tasks and parse the data. A proof of concept perl script can be found at http://www.moonpie.org/tools/discover.tgz

4. Conclusion

Hopefully this paper has provided some useful information and has caused you to question, or more closely examine your methodology for discovering hosts. It should also serve as an awareness guide for firewall administrators in how to develop explicit rules to hide Internet accessible hosts as much as possible.

Many thanks go to Fyodor for his efforts in creating the most powerful open source port scanner, nmap. More information on nmap can be found at http://www.nmap.org and http://www.insecure.org.