The Science of Safely Finding an Unused IP Address

During pentests you're often allocated an IP by the client or can get one via DHCP. There are times, however when the client might expect you find a free IP on your own. Or you might want to check that the client hasn't assigned you an IP address that's already in use.

I'm sure we've all got our own techniques for doing this. Each will have a greater or lesser risk of causing a temporary IP clash. This post explores some ways that we can minimise the risk picking an IP address that's in use.

I make heavy use of arp-scan during the explanation below and also use the fingerprint database of arp-fingerprint. So, thanks upfront to arp-scan's author, Roy Hills for a very useful tool and comprehensive database that made this blog post much quicker to write than it otherwise would have been. That said, this post isn't really about tools, just a methodology for finding a free IP address.

Network Sniffing

If you find yourself on a network and DHCP fails, your best course of action is probably to sniff on the network for a while (e.g. with tcpdump or wireshark). Hopefully you'll seem some broadcast traffic that will give you an idea of some of the IP addresses in use.

For the sake of an example, let's assume we've seen traffic from 10.0.0.1.

Guess a Network Range

Now guess a netmask in which you'll search for a free IP address – you can always expand the network range later when you've found a free IP and want to start your pentest. Perhaps start by assuming a Class C network, so we're looking to find the free IPs in 10.0.0.0/24.

Use ARP Queries to Identify IPs in Use

This is the main point of the post. It seems that a few well chose ARP requests will mean that your probing is both effective and minimises the chances of causing an IP clash.

We know that we need to scan 10.0.0.0/24, but what should we choose as our source IP address? Arp-fingerprint's database indicates that the following would be good choices:

- 127.0.0.1
- 0.0.0.0
- 255.255.255.255
- 1.0.0.1 (IP network 1.0.0.0/8 is reserved by IANA)

So the corresponding arp-scan commands would be:

```
arp-scan --arpspa=127.0.0.1 10.0.0.0/24
arp-scan --arpspa=0.0.0.0 10.0.0.0/24
arp-scan --arpspa=255.255.255.255 10.0.0.0/24
arp-scan --arpspa=1.0.0.1 10.0.0.0/24
```

But how effective is this going to be? Do systems generally respond to at least one of these probes? According to arp-fingerprint's database, most OSs we're likely to encounter will respond. Below is an extract from arp-fingerprint's database (arp-scan v1.8.1). The OSs below where one of the first 4 digits is a "1" should be detected:

```
my %fp_hash = (
'11110100000' => 'FreeBSD 5.3, 7.0, DragonflyBSD 2.0, Win98, WinME, NT4, 2000, XP, 2003, Catalyst IOS 12.0, 12.1, 12.2, FortiOS 3.00',
'01000100000' => 'Linux 2.2, 2.4, 2.6',
'01010100000' => 'Linux 2.2, 2.4, 2.6, Vista, 2008, Windows7', # Linux only if non-local IP is routed
'00000100000' => 'Cisco IOS 11.2, 11.3, 12.0, 12.1, 12.2, 12.3, 12.4',
```

```
'11110110000' => 'Solaris 2.5.1, 2.6, 7, 8, 9, 10, HP-UX 11',
'01000111111' => 'ScreenOS 5.0, 5.1, 5.3, 5.4',
'11110000000' => 'Linux 2.0, MacOS 10.4, IPSO 3.2.1, Minix 3, Cisco VPN Concentrator 4.7, Catalyst 1900',
'11110100011' => 'MacOS 10.3, FreeBSD 4.3, IRIX 6.5, AIX 4.3, AIX 5.3',
'10010100011' => 'SCO OS 5.0.7',
'10110100000' => 'Win 3.11, 95, NT 3.51',
'11110000011' => '2.11BSD, 4.3BSD, OpenBSD 3.1, OpenBSD 3.9, Nortel Contivity 6.00, 6.05',
'10110110000' => 'NetBSD 2.0.2, 4.0',
'10110111111' => 'PIX OS 4.4, 5.1, 5.2, 5.3',
'11110111111' => 'PIX OS 6.0, 6.1, 6.2, ScreenOS 5.0 (transparent), Plan9, Blackberry OS',
'00010110011' => 'PIX OS 6.3, 7.0(1), 7.0(2)',
'01010110011' => 'PIX OS 7.0(4)-7.0(6), 7.1, 7.2, 8.0',
'00000110000' => 'Netware 6.5',
'00010100000' => 'Unknown 1', # 14805 79.253 Cisco
'00000110011' => 'Cisco IP Phone 79xx SIP 5.x,6.x,7.x',
'11110110011' => 'Cisco IP Phone 79xx SIP 8.x', # Also 14805 63.11 Fujitsu Siemens
);
```

So by covering pretty much every version of Windows, Linux and Solaris, we've covered most of the servers and workstations types we're likely to encounter on pentests (or at least the main ones that I encounter). The following won't be found:

```
'00000100000' => 'Cisco IOS 11.2, 11.3, 12.0, 12.1, 12.2, 12.3, 12.4',
'00000110000' => 'Netware 6.5',
'00000110011' => 'Cisco IP Phone 79xx SIP 5.x,6.x,7.x',
```

The Risky Bit

So we think we've found most of the IPs that are used in 10.0.0.0/24, but we're not absolutely sure.

Now, we can now do a small number of ARP probes from what we think is an unused IP address in the range. In fact, we'll pick two IP addresses so we can verify that they're both really free.

1. Choose two (apparently) free addresses between the smallest and largest IP you've observed. Don't choose an IP outside of this range unless you have to because your guess at the netmask might have been wrong. We'll choose 10.0.0.99 and 10.0.0.11 for this example.
2. From each IP address, make an ARP request for the other

```
arp-scan --arpspa=10.0.0.11 10.0.0.99
arp-scan --arpspa=10.0.0.99 10.0.0.11
```

If you receive no response these probes, you can be sure they're both free. Pick one and do an "arp-scan -l" with various netmasks until you're happy your netmask is big enough.

If you receive a response to one of the probes above, change the used IP for another apprently free one and repeat.

Disclaimer

To the best of my knowledge the requests recommended above should be relatively safe or at least show diligence on your part. I accept no responsibility if it anything goes wrong, though.

ipstackquirks

Blog

timing-attack-checker
gateway-finder

Categories

- Blog (78)
- Cheat Sheets (10)
  - Shells (1)
  - SQL Injection (7)
- Contact (2)
- Site News (3)

- Tools (17)
  - Audit (3)
  - Misc (7)
  - User Enumeration (4)
  - Web Shells (3)
- Uncategorized (3)
- Yaptest (15)
  - Front End (1)
  - Installing (2)
  - Overview (2)
  - Using (8)