**Chapter2 : Network Mapper**

1: Map the Network1

Nmap (“Network Mapper”) is an open source tool for network exploration and security auditing. It was designed to rapidly scan large networks, although it works fine against single hosts. Nmap uses raw IP packets in novel ways to determine what hosts are available on the network, what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics. While Nmap is commonly used for security audits, many systems and network administrators find it useful for routine tasks such as network inventory, managing service upgrade schedules, and monitoring host or service uptime.

The output from Nmap is a list of scanned targets, with supplemental information on each depending on the options used. Key among that information is the “interesting ports table”. That table lists the port number and protocol, service name, and state. The state is either open, filtered, closed, or unfiltered. Open means that an application on the target machine is listening for connections/packets on that port. Filtered means that a firewall, filter, or other network obstacle is blocking the port so that Nmap cannot tell whether it is open or closed. Closed ports have no application listening on them, though they could open up at any time. Ports are classified as unfiltered when they are responsive to Nmap's probes, but Nmap cannot determine whether they are open or closed. Nmap reports the state combinations open|filtered and closed|filtered when it cannot determine which of the two states describe a port. The port table may also include software version details when version detection has been requested. When an IP protocol scan is requested (-sO), Nmap provides information on supported IP protocols rather than listening ports.

In addition to the interesting ports table, Nmap can provide further information on targets, including reverse DNS names, operating system guesses, device types, and MAC addresses.

A typical Nmap scan is shown in [Example 15.1](https://nmap.org/book/man.html#man-ex-repscan). The only Nmap arguments used in this example are -A, to enable OS and version detection, script scanning, and traceroute; -T4 for faster execution; and then the hostname.

# **nmap -A -T4 scanme.nmap.org**

Nmap scan report for scanme.nmap.org (74.207.244.221)

Host is up (0.029s latency).

rDNS record for 74.207.244.221: li86-221.members.linode.com

Not shown: 995 closed ports

PORT STATE SERVICE VERSION

22/tcp open ssh OpenSSH 5.3p1 Debian 3ubuntu7 (protocol 2.0)

| ssh-hostkey: 1024 8d:60:f1:7c:ca:b7:3d:0a:d6:67:54:9d:69:d9:b9:dd (DSA)

|\_2048 79:f8:09:ac:d4:e2:32:42:10:49:d3:bd:20:82:85:ec (RSA)

80/tcp open http Apache httpd 2.2.14 ((Ubuntu))

|\_http-title: Go ahead and ScanMe!

646/tcp filtered ldp

1720/tcp filtered H.323/Q.931

9929/tcp open nping-echo Nping echo

Device type: general purpose

Running: Linux 2.6.X

OS CPE: cpe:/o:linux:linux\_kernel:2.6.39

OS details: Linux 2.6.39

Network Distance: 11 hops

Service Info: OS: Linux; CPE: cpe:/o:linux:kernel

TRACEROUTE (using port 53/tcp)

HOP RTT ADDRESS

[Cut first 10 hops for brevity]

11 17.65 ms li86-221.members.linode.com (74.207.244.221)

Nmap done: 1 IP address (1 host up) scanned in 14.40 seconds

The newest version of Nmap can be obtained from [https://nmap.org](https://nmap.org/). The newest version of this man page is available at <https://nmap.org/book/man.html>. It is also included as a chapter of Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning.

[http://172.30.2.67:8000](http://172.30.2.67:8000/)

[http://172.30.2.67:8001](http://172.30.2.67:8001/)

<http://172.30.2.67:8003>

2 ; Map the Network2

Everything on the Nmap command-line that isn't an option (or option argument) is treated as a target host specification. The simplest case is to specify a target IP address or hostname for scanning.

When a hostname is given as a target, it is *resolved* via the Domain Name System (DNS) to determine the IP address to scan. If the name resolves to more than one IP address, only the first one will be scanned. To make Nmap scan all the resolved addresses instead of only the first one, use the --resolve-all option.

Sometimes you wish to scan a whole network of adjacent hosts. For this, Nmap supports CIDR-style addressing. You can append /*<numbits>* to an IP address or hostname and Nmap will scan every IP address for which the first *<numbits>* are the same as for the reference IP or hostname given. For example, 192.168.10.0/24 would scan the 256 hosts between 192.168.10.0 (binary: 11000000 10101000 00001010 00000000) and 192.168.10.255 (binary: 11000000 10101000 00001010 11111111), inclusive. 192.168.10.40/24 would scan exactly the same targets. Given that the host scanme.nmap.org is at the IP address 64.13.134.52, the specification scanme.nmap.org/16 would scan the 65,536 IP addresses between 64.13.0.0 and 64.13.255.255. The smallest allowed value is /0, which targets the whole Internet. The largest value for IPv4 is /32, which scans just the named host or IP address because all address bits are fixed. The largest value for IPv6 is /128, which does the same thing.

CIDR notation is short but not always flexible enough. For example, you might want to scan 192.168.0.0/16 but skip any IPs ending with .0 or .255 because they may be used as subnet network and broadcast addresses. Nmap supports this through octet range addressing. Rather than specify a normal IP address, you can specify a comma-separated list of numbers or ranges for each octet. For example, 192.168.0-255.1-254 will skip all addresses in the range that end in .0 or .255, and 192.168.3-5,7.1 will scan the four addresses 192.168.3.1, 192.168.4.1, 192.168.5.1, and 192.168.7.1. Either side of a range may be omitted; the default values are 0 on the left and 255 on the right. Using - by itself is the same as 0-255, but remember to use 0- in the first octet so the target specification doesn't look like a command-line option. Ranges need not be limited to the final octets: the specifier 0-255.0-255.13.37 will perform an Internet-wide scan for all IP addresses ending in 13.37. This sort of broad sampling can be useful for Internet surveys and research.

IPv6 addresses can be specified by their fully qualified IPv6 address or hostname or with CIDR notation for subnets. Octet ranges aren't yet supported for IPv6.

IPv6 addresses with non-global scope need to have a zone ID suffix. On Unix systems, this is a percent sign followed by an interface name; a complete address might be fe80::a8bb:ccff:fedd:eeff%eth0. On Windows, use an interface index number in place of an interface name: fe80::a8bb:ccff:fedd:eeff%1. You can see a list of interface indexes by running the command **netsh.exe interface ipv6 show interface**.

Nmap accepts multiple host specifications on the command line, and they don't need to be the same type. The command **nmap scanme.nmap.org 192.168.0.0/8 10.0.0,1,3-7.-** does what you would expect.

While targets are usually specified on the command lines, the following options are also available to control target selection:

-iL *<inputfilename>* (Input from list)

Reads target specifications from *<inputfilename>*. Passing a huge list of hosts is often awkward on the command line, yet it is a common desire. For example, your DHCP server might export a list of 10,000 current leases that you wish to scan. Or maybe you want to scan all IP addresses *except* for those to locate hosts using unauthorized static IP addresses. Simply generate the list of hosts to scan and pass that filename to Nmap as an argument to the -iL option. Entries can be in any of the formats accepted by Nmap on the command line (IP address, hostname, CIDR, IPv6, or octet ranges). Each entry must be separated by one or more spaces, tabs, or newlines. You can specify a hyphen (-) as the filename if you want Nmap to read hosts from standard input rather than an actual file.

The input file may contain comments that start with # and extend to the end of the line.

-iR *<num hosts>* (Choose random targets)

For Internet-wide surveys and other research, you may want to choose targets at random. The *<num hosts>* argument tells Nmap how many IPs to generate. Undesirable IPs such as those in certain private, multicast, or unallocated address ranges are automatically skipped. The argument 0 can be specified for a never-ending scan. Keep in mind that some network administrators bristle at unauthorized scans of their networks and may complain. Use this option at your own risk! If you find yourself really bored one rainy afternoon, try the command **nmap -Pn -sS -p 80 -iR 0 --open** to locate random web servers for browsing.

--exclude *<host1>*[,*<host2>*[,...]] (Exclude hosts/networks)

Specifies a comma-separated list of targets to be excluded from the scan even if they are part of the overall network range you specify. The list you pass in uses normal Nmap syntax, so it can include hostnames, CIDR netblocks, octet ranges, etc. This can be useful when the network you wish to scan includes untouchable mission-critical servers, systems that are known to react adversely to port scans, or subnets administered by other people.

--excludefile *<exclude\_file>* (Exclude list from file)

This offers the same functionality as the --exclude option, except that the excluded targets are provided in a newline-, space-, or tab-delimited *<exclude\_file>* rather than on the command line.

The exclude file may contain comments that start with # and extend to the end of the line.

-n (No DNS resolution)

Tells Nmap to *never* do reverse DNS resolution on the active IP addresses it finds. Since DNS can be slow even with Nmap's built-in parallel stub resolver, this option can slash scanning times.

-R (DNS resolution for all targets)

Tells Nmap to *always* do reverse DNS resolution on the target IP addresses. Normally reverse DNS is only performed against responsive (online) hosts.

--resolve-all (Scan each resolved address)

If a hostname target resolves to more than one address, scan all of them. The default behavior is to only scan the first resolved address. Regardless, only addresses in the appropriate address family will be scanned: IPv4 by default, IPv6 with -6.

--unique (Scan each address only once)

Scan each IP address only once. The default behavior is to scan each address as many times as it is specified in the target list, such as when network ranges overlap or different hostnames resolve to the same address.

--system-dns (Use system DNS resolver)

By default, Nmap reverse-resolves IP addresses by sending queries directly to the name servers configured on your host and then listening for responses. Many requests (often dozens) are performed in parallel to improve performance. Specify this option to use your system resolver instead (one IP at a time via the getnameinfo call). This is slower and rarely useful unless you find a bug in the Nmap parallel resolver (please let us know if you do). The system resolver is always used for forward lookups (getting an IP address from a hostname).

--dns-servers *<server1>*[,*<server2>*[,...]] (Servers to use for reverse DNS queries)

By default, Nmap determines your DNS servers (for rDNS resolution) from your resolv.conf file (Unix) or the Registry (Win32). Alternatively, you may use this option to specify alternate servers. This option is not honored if you are using --system-dns. Using multiple DNS servers is often faster, especially if you choose authoritative servers for your target IP space. This option can also improve stealth, as your requests can be bounced off just about any recursive DNS server on the Internet.

This option also comes in handy when scanning private networks. Sometimes only a few name servers provide proper rDNS information, and you may not even know where they are. You can scan the network for port 53 (perhaps with version detection), then try Nmap list scans (-sL) specifying each name server one at a time with --dns-servers until you find one which works.

This option might not be honored if the DNS response exceeds the size of a UDP packet. In such a situation our DNS resolver will make the best effort to extract a response from the truncated packet, and if not successful it will fall back to using the system resolver. Also, responses that contain CNAME aliases will fall back to the system resolver.

[http://172.30.2.67:8000](http://172.30.2.67:8000/)

[http://172.30.2.67:8001](http://172.30.2.67:8001/)

<http://172.30.2.67:8003>

3 :Map the Network 3 Scripting Engine

The Nmap Scripting Engine (NSE) is one of Nmap's most powerful and flexible features. It allows users to write (and share) simple scripts (using the [Lua programming language](http://lua.org/" \t "_top) ) to automate a wide variety of networking tasks. Those scripts are executed in parallel with the speed and efficiency you expect from Nmap. Users can rely on the growing and diverse set of scripts distributed with Nmap, or write their own to meet custom needs.

Tasks we had in mind when creating the system include network discovery, more sophisticated version detection, vulnerability detection. NSE can even be used for vulnerability exploitation.

To reflect those different uses and to simplify the choice of which scripts to run, each script contains a field associating it with one or more categories. Currently defined categories are auth, broadcast, default. discovery, dos, exploit, external, fuzzer, intrusive, malware, safe, version, and vuln. These are all described in [the section called “Script Categories”](https://nmap.org/book/nse-usage.html#nse-categories).

Scripts are not run in a sandbox and thus could accidentally or maliciously damage your system or invade your privacy. Never run scripts from third parties unless you trust the authors or have carefully audited the scripts yourself.

The Nmap Scripting Engine is described in detail in [Chapter 9, *Nmap Scripting Engine*](https://nmap.org/book/nse.html) and is controlled by the following options:

-sC

Performs a script scan using the default set of scripts. It is equivalent to --script=default. Some of the scripts in this category are considered intrusive and should not be run against a target network without permission.

--script *<filename>*|*<category>*|*<directory>*/|*<expression>*[,...]

Runs a script scan using the comma-separated list of filenames, script categories, and directories. Each element in the list may also be a Boolean expression describing a more complex set of scripts. Each element is interpreted first as an expression, then as a category, and finally as a file or directory name.

There are two special features for advanced users only. One is to prefix script names and expressions with + to force them to run even if they normally wouldn't (e.g. the relevant service wasn't detected on the target port). The other is that the argument all may be used to specify every script in Nmap's database. Be cautious with this because NSE contains dangerous scripts such as exploits, brute force authentication crackers, and denial of service attacks.

File and directory names may be relative or absolute. Absolute names are used directly. Relative paths are looked for in the scripts of each of the following places until found:

|  |
| --- |
| --datadir |
| $NMAPDIR |
| ~/.nmap (not searched on Windows) |
| *<APPDATA>*\nmap (only on Windows) |
| the directory containing the nmap executable |
| the directory containing the nmap executable, followed by ../share/nmap (not searched on Windows) |
| NMAPDATADIR (not searched on Windows) |
| the current directory. |

When a directory name ending in / is given, Nmap loads every file in the directory whose name ends with .nse. All other files are ignored and directories are not searched recursively. When a filename is given, it does not have to have the .nse extension; it will be added automatically if necessary.

Nmap scripts are stored in a scripts subdirectory of the Nmap data directory by default (see [Chapter 14, *Understanding and Customizing Nmap Data Files*](https://nmap.org/book/data-files.html)). For efficiency, scripts are indexed in a database stored in scripts/script.db, which lists the category or categories in which each script belongs.

When referring to scripts from script.db by name, you can use a shell-style ‘\*’ wildcard.

**nmap --script "http-\*"**

Loads all scripts whose name starts with http-, such as http-auth and http-open-proxy. The argument to --script had to be in quotes to protect the wildcard from the shell.

More complicated script selection can be done using the and, or, and not operators to build Boolean expressions. The operators have the same [precedence](http://www.lua.org/manual/5.1/manual.html#2.5.3) as in Lua: not is the highest, followed by and and then or. You can alter precedence by using parentheses. Because expressions contain space characters it is necessary to quote them.

**nmap --script "not intrusive"**

Loads every script except for those in the intrusive category.

**nmap --script "default or safe"**

This is functionally equivalent to **nmap --script "default,safe"**. It loads all scripts that are in the default category or the safe category or both.

**nmap --script "default and safe"**

Loads those scripts that are in *both* the default and safe categories.

**nmap --script "(default or safe or intrusive) and not http-\*"**

Loads scripts in the default, safe, or intrusive categories, except for those whose names start with http-.

--script-args *<n1>*=*<v1>*,*<n2>*={*<n3>*=*<v3>*},*<n4>*={*<v4>*,*<v5>*}

Lets you provide arguments to NSE scripts. Arguments are a comma-separated list of name=value pairs. Names and values may be strings not containing whitespace or the characters ‘{’, ‘}’, ‘=’, or ‘,’. To include one of these characters in a string, enclose the string in single or double quotes. Within a quoted string, ‘\’ escapes a quote. A backslash is only used to escape quotation marks in this special case; in all other cases a backslash is interpreted literally. Values may also be tables enclosed in {}, just as in Lua. A table may contain simple string values or more name-value pairs, including nested tables. Many scripts qualify their arguments with the script name, as in xmpp-info.server\_name. You may use that full qualified version to affect just the specified script, or you may pass the unqualified version (server\_name in this case) to affect all scripts using that argument name. A script will first check for its fully qualified argument name (the name specified in its documentation) before it accepts an unqualified argument name. A complex example of script arguments is --script-args 'user=foo,pass=",{}=bar",whois={whodb=nofollow+ripe},xmpp-info.server\_name=localhost'. The online NSE Documentation Portal at <https://nmap.org/nsedoc/> lists the arguments that each script accepts.

--script-args-file *<filename>*

Lets you load arguments to NSE scripts from a file. Any arguments on the command line supersede ones in the file. The file can be an absolute path, or a path relative to Nmap's usual search path (NMAPDIR, etc.) Arguments can be comma-separated or newline-separated, but otherwise follow the same rules as for --script-args, without requiring special quoting and escaping, since they are not parsed by the shell.

--script-help *<filename>*|*<category>*|*<directory>*|*<expression>*|all[,...]

Shows help about scripts. For each script matching the given specification, Nmap prints the script name, its categories, and its description. The specifications are the same as those accepted by --script; so for example if you want help about the ftp-anon script, you would run **nmap --script-help ftp-anon**. In addition to getting help for individual scripts, you can use this as a preview of what scripts will be run for a specification, for example with **nmap --script-help default**.

--script-trace

This option does what --packet-trace does, just one ISO layer higher. If this option is specified all incoming and outgoing communication performed by a script is printed. The displayed information includes the communication protocol, the source, the target and the transmitted data. If more than 5% of all transmitted data is not printable, then the trace output is in a hex dump format. Specifying --packet-trace enables script tracing too.

--script-updatedb

This option updates the script database found in scripts/script.db which is used by Nmap to determine the available default scripts and categories. It is only necessary to update the database if you have added or removed NSE scripts from the default scripts directory or if you have changed the categories of any script. This option is generally used by itself: **nmap --script-updatedb**.

[http://172.30.2.67:8000](http://172.30.2.67:8000/)

[http://172.30.2.67:8001](http://172.30.2.67:8001/)

<http://172.30.2.67:8003>

4 :Firewall Detection

The first step toward bypassing firewall rules is to understand them. Where possible, Nmap distinguishes between ports that are reachable but closed, and those that are actively filtered. An effective technique is to start with a normal SYN port scan, then move on to more exotic techniques such as ACK scan and IP ID sequencing to gain a better understanding of the network.

### Standard SYN Scan

One helpful feature of the TCP protocol is that systems are required by [RFC 793](http://www.rfc-editor.org/rfc/rfc793.txt) to send a negative response to unexpected connection requests in the form of a TCP RST (reset) packet. The RST packet makes closed ports easy for Nmap to recognize. Filtering devices such as firewalls, on the other hand, tend to drop packets destined for disallowed ports. In some cases they send ICMP error messages (usually port unreachable) instead. Because dropped packets and ICMP errors are easily distinguishable from RST packets, Nmap can reliably detect filtered TCP ports from open or closed ones, and it does so automatically. This is shown in [Example 10.1](https://nmap.org/book/determining-firewall-rules.html#defeating-firewalls-ids-standardsyn).

Example 10.1. Detection of closed and filtered TCP ports

# **nmap -sS -T4 scanme.nmap.org**

Starting Nmap ( https://nmap.org )

Nmap scan report for scanme.nmap.org (64.13.134.52)

Not shown: 994 filtered ports

PORT STATE SERVICE

22/tcp open ssh

25/tcp closed smtp

53/tcp open domain

70/tcp closed gopher

80/tcp open http

113/tcp closed auth

Nmap done: 1 IP address (1 host up) scanned in 5.40 seconds

One of the most important lines in [Example 10.1](https://nmap.org/book/determining-firewall-rules.html#defeating-firewalls-ids-standardsyn) is the note “Not shown: 994 filtered ports”. In other words, this host has a proper deny-by-default firewall policy. Only those ports the administrator explicitly allowed are reachable, while the default action is to deny (filter) them. Three of the enumerated ports are in the open state (22, 53, and 80), and another three are closed (25, 70, and 113). The remaining 994 tested ports are unreachable by this standard scan (filtered).

#### Sneaky firewalls that return RST

While the Nmap distinction between closed TCP ports (which return a RST packet) and filtered ports (returning nothing or an ICMP error) is usually accurate, many firewall devices are now capable of forging RST packets as though they are coming from the destination host and claiming that the port is closed. One example of this capability is the Linux iptables system, which offers many methods for rejecting undesired packets. The iptables man page documents this feature as follows:

--reject-with type

The type given can be icmp-net-unreachable, icmp-host-unreachable, icmp-port-unreachable, icmp-proto-unreachable, icmp-net-prohibited or icmp-host-prohibited, which return the appropriate ICMP error message (port-unreachable is the default). The option tcp-reset can be used on rules which only match the TCP protocol: this causes a TCP RST packet to be sent back. This is mainly useful for blocking ident (113/tcp) probes which frequently occur when sending mail to broken mail hosts (which won't accept your mail otherwise).

Forging RST packets by firewalls and IDS/IPS is not particularly common outside of port 113, as it can be confusing to legitimate network operators and it also allows scanners to move on to the next port immediately without waiting on the timeout caused by dropped packets. Nevertheless, it does happen. Such forgery can usually be detected by careful analysis of the RST packet in comparison with other packets sent by the machine. [the section called “Detecting Packet Forgery by Firewall and Intrusion Detection Systems”](https://nmap.org/book/firewall-ids-packet-forgery.html) describes effective techniques for doing so.

### ACK Scan

As described in depth in [the section called “TCP ACK Scan (-sA)”](https://nmap.org/book/scan-methods-ack-scan.html), the ACK scan sends TCP packets with only the ACK bit set. Whether ports are open or closed, the target is required by [RFC 793](http://www.rfc-editor.org/rfc/rfc793.txt) to respond with a RST packet. Firewalls that block the probe, on the other hand, usually make no response or send back an ICMP destination unreachable error. This distinction allows Nmap to report whether the ACK packets are being filtered. The set of filtered ports reported by an Nmap ACK scan is often smaller than for a SYN scan against the same machine because ACK scans are more difficult to filter. Many networks allow nearly unrestricted outbound connections, but wish to block Internet hosts from initiating connections back to them. Blocking incoming SYN packets (without the ACK bit set) is an easy way to do this, but it still allows any ACK packets through. Blocking those ACK packets is more difficult, because they do not tell which side started the connection. To block unsolicited ACK packets (as sent by the Nmap ACK scan), while allowing ACK packets belonging to legitimate connections, firewalls must statefully watch every established connection to determine whether a given ACK is appropriate. These stateful firewalls are usually more secure because they can be more restrictive. Blocking ACK scans is one extra available restriction. The downsides are that they require more resources to function, and a stateful firewall reboot can cause a device to lose state and terminate all established connections passing through it.

While stateful firewalls are widespread and rising in popularity, the stateless approach is still quite common. For example, the Linux Netfilter/iptables system supports the --syn convenience option to make the stateless approach described above easy to implement.

In the previous section, a SYN scan showed that all but six of 1,000 common ports on scanme.nmap.org were in the filtered state. [Example 10.2](https://nmap.org/book/determining-firewall-rules.html#defeating-firewalls-ids-ackscan-scanme) demonstrates an ACK scan against the same host to determine whether it is using a stateful firewall.

Example 10.2. ACK scan against Scanme

# **nmap -sA -T4 scanme.nmap.org**

Starting Nmap ( https://nmap.org )

Nmap scan report for scanme.nmap.org (64.13.134.52)

Not shown: 994 filtered ports

PORT STATE SERVICE

22/tcp unfiltered ssh

25/tcp unfiltered smtp

53/tcp unfiltered domain

70/tcp unfiltered gopher

80/tcp unfiltered http

113/tcp unfiltered auth

Nmap done: 1 IP address (1 host up) scanned in 5.96 seconds

The same six ports displayed in the SYN scan are shown here. The other 994 are still filtered. This is because Scanme is protected by this stateful iptables directive: **iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT**. This only accepts packets that are part of or related to an established connection. Unsolicited ACK packets sent by Nmap are dropped, except to the six special ports shown. Special rules allow all packets to the ports 22, 25, 53, 70, and 80, as well as sending a RST packet in response to port 113 probes. Note that the six shown ports are in the unfiltered state, since the ACK scan cannot further divide them into open (22, 53, and 80) or closed (25, 70, 113).

Now let us look at another example. A Linux host named Para on my local network uses the following (simplified to save space) firewall script:

#!/bin/sh

#

# A simple, stateless, host-based firewall script.

# First of all, flush & delete any existing tables

iptables -F

iptables -X

# Deny by default (input/forward)

iptables --policy INPUT DROP

iptables --policy OUTPUT ACCEPT

iptables --policy FORWARD DROP

# I want to make ssh and www accessible from outside

iptables -A INPUT -m multiport -p tcp --destination-port 22,80 -j ACCEPT

# Allow responses to outgoing TCP requests

iptables -A INPUT --proto tcp ! --syn -j ACCEPT

This firewall is stateless, as there is no sign of the --state option or the -m state module request. [Example 10.3](https://nmap.org/book/determining-firewall-rules.html#defeating-firewalls-ids-scans-para) shows SYN and ACK scans against this host.

Example 10.3. Contrasting SYN and ACK scans against Para

# **nmap -sS -p1-100 -T4 para**

Starting Nmap ( https://nmap.org )

Nmap scan report for para (192.168.10.191)

Not shown: 98 filtered ports

PORT STATE SERVICE

22/tcp open ssh

80/tcp closed http

MAC Address: 00:60:1D:38:32:90 (Lucent Technologies)

Nmap done: 1 IP address (1 host up) scanned in 3.81 seconds

# **nmap -sA -p1-100 -T4 para**

Starting Nmap ( https://nmap.org )

All 100 scanned ports on para (192.168.10.191) are: unfiltered

MAC Address: 00:60:1D:38:32:90 (Lucent Technologies)

Nmap done: 1 IP address (1 host up) scanned in 0.70 seconds

In the SYN scan, 98 of 100 ports are filtered. Yet the ACK scan shows every scanned port being unfiltered. In other words, all of the ACK packets are sneaking through unhindered and eliciting RST responses. These responses also make the scan more than five times as fast, since it does not have to wait on timeouts.

Now we know how to distinguish between stateful and stateless firewalls, but what good is that? The ACK scan of Para shows that some packets are probably reaching the destination host. I say probably because firewall forgery is always possible. While you may not be able to establish TCP connections to those ports, they can be useful for determining which IP addresses are in use, OS detection tests, certain IP ID shenanigans, and as a channel for tunneling commands to rootkits installed on those machines. Other scan types, such as FIN scan, may even be able to determine which ports are open and thus infer the purpose of the hosts. Such hosts may be useful as zombies for an IP ID idle scan.

This pair of scans also demonstrates that what we are calling a port state is not solely a property of the port itself. Here, the same port number is considered filtered by one scan type and unfiltered by another. What IP address you scan from, the rules of any filtering devices along the way, and which interface of the target machine you access can all affect how Nmap sees the ports. The port table only reflects what Nmap saw when running from a particular machine, with a defined set of options, at the given time.

### IP ID Tricks

The humble identification field within IP headers can divulge a surprising amount of information. Later in this chapter it will be used for port scanning (idle scan technique) and to detect when firewall and intrusion detection systems are forging RST packets as though they come from protected hosts. Another neat trick is to discern what source addresses make it through the firewall. There is no point spending hours on a blind spoofing attack “from” 192.168.0.1 if some firewall along the way drops all such packets.

I usually test this condition with Nping, the free network probing tool that comes with Nmap. This is a rather complex technique, but it can be valuable sometimes. Here are the steps I take:

1. Find at least one accessible (open or closed) port of one machine on the internal network. Routers, printers, and Windows boxes often work well. Recent releases of Linux, Solaris, and OpenBSD have largely resolved the issue of predictable IP ID sequence numbers and will not work. The machine chosen should have little network traffic to avoid confusing results.
2. Verify that the machine has predictable IP ID sequences. The following command tests a Windows XP machine named Playground. The Nping options request that five SYN packets be sent to port 80, one second apart.
3. # **nping -c 5 --delay 1 -p 80 --tcp playground**
4. Starting Nping ( https://nmap.org/nping )
5. SENT (0.0210s) TCP 192.168.0.21:42091 > 192.168.0.40:80 S ttl=64 id=48089iplen=40 seq=136013019 win=1480
6. RCVD (0.0210s) TCP 192.168.0.40:80 > 192.168.0.21:42091 RA ttl=128 id=**4900**iplen=40 seq=0 win=0
7. SENT (1.0220s) TCP 192.168.0.21:42091 > 192.168.0.40:80 S ttl=64 id=41250 iplen=40 seq=136013019 win=1480
8. RCVD (1.0220s) TCP 192.168.0.40:80 > 192.168.0.21:42091 RA ttl=128 id=**4901**iplen=40 seq=0 win=0
9. SENT (2.0240s) TCP 192.168.0.21:42091 > 192.168.0.40:80 S ttl=64 id=10588 iplen=40 seq=136013019 win=1480
10. RCVD (2.0250s) TCP 192.168.0.40:80 > 192.168.0.21:42091 RA ttl=128 id=**4902**iplen=40 seq=0 win=0
11. SENT (3.0270s) TCP 192.168.0.21:42091 > 192.168.0.40:80 S ttl=64 id=55928 iplen=40 seq=136013019 win=1480
12. RCVD (3.0280s) TCP 192.168.0.40:80 > 192.168.0.21:42091 RA ttl=128 id=**4903**iplen=40 seq=0 win=0
13. SENT (4.0300s) TCP 192.168.0.21:42091 > 192.168.0.40:80 S ttl=64 id=3309 iplen=40 seq=136013019 win=1480
14. RCVD (4.0300s) TCP 192.168.0.40:80 > 192.168.0.21:42091 RA ttl=128 id=**4904**iplen=40 seq=0 win=0
15. Max rtt: 0.329ms | Min rtt: 0.288ms | Avg rtt: 0.300ms
16. Raw packets sent: 5 (200B) | Rcvd: 5 (230B) | Lost: 0 (0.00%)
17. Tx time: 4.00962s | Tx bytes/s: 49.88 | Tx pkts/s: 1.25
18. Rx time: 5.01215s | Rx bytes/s: 45.89 | Rx pkts/s: 1.00
19. Nping done: 1 IP address pinged in 5.03 seconds

Since the IP ID fields are perfectly sequential, we can move on to the next test. If they were random or very far apart, we would have to find a new accessible host.

1. Start a flood of probes to the target from a host near your own (just about any host will do). An example command is **nping -S scanme.nmap.org --rate 10 -p 80 -c 10000 --tcp playground**. Replace scanme.nmap.org with some other host of your choice, and playground with your target host. Getting replies back is not necessary, because the goal is simply to increment the IP ID sequences. Do not use the real address of the machine you are running Nping from. Using a machine nearby on the network is advised to reduce the probability that your own ISP will block the packets.

While this is going on, redo the test from the previous step against your target machine.

# **nping -c 5 --delay 1 -p 80 --tcp playground**

Starting Nping ( https://nmap.org/nping )

SENT (0.0210s) TCP 192.168.0.21:1781 > 192.168.0.40:80 S ttl=64 id=61263iplen=40 seq=292367194 win=1480

RCVD (0.0220s) TCP 192.168.0.40:80 > 192.168.0.21:1781 RA ttl=128 id=**5755**iplen=40 seq=0 win=0

SENT (1.0220s) TCP 192.168.0.21:1781 > 192.168.0.40:80 S ttl=64 id=30096iplen=40 seq=292367194 win=1480

RCVD (1.0220s) TCP 192.168.0.40:80 > 192.168.0.21:1781 RA ttl=128 id=**5766**iplen=40 seq=0 win=0

SENT (2.0240s) TCP 192.168.0.21:1781 > 192.168.0.40:80 S ttl=64 id=26815iplen=40 seq=292367194 win=1480

RCVD (2.0240s) TCP 192.168.0.40:80 > 192.168.0.21:1781 RA ttl=128 id=**5777**iplen=40 seq=0 win=0

SENT (3.0260s) TCP 192.168.0.21:1781 > 192.168.0.40:80 S ttl=64 id=49116iplen=40 seq=292367194 win=1480

RCVD (3.0270s) TCP 192.168.0.40:80 > 192.168.0.21:1781 RA ttl=128 id=**5788**iplen=40 seq=0 win=0

SENT (4.0290s) TCP 192.168.0.21:1781 > 192.168.0.40:80 S ttl=64 id=2916iplen=40 seq=292367194 win=1480

RCVD (4.0300s) TCP 192.168.0.40:80 > 192.168.0.21:1781 RA ttl=128 id=**5799**iplen=40 seq=0 win=0

Max rtt: 0.342ms | Min rtt: 0.242ms | Avg rtt: 0.272ms

Raw packets sent: 5 (200B) | Rcvd: 5 (230B) | Lost: 0 (0.00%)

Tx time: 4.00853s | Tx bytes/s: 49.89 | Tx pkts/s: 1.25

Rx time: 5.01106s | Rx bytes/s: 45.90 | Rx pkts/s: 1.00

Nping done: 1 IP address pinged in 5.03 seconds

This time, the IP IDs are increasing by roughly 11 per second instead of one. The target is receiving our 10 forged packets per second, and responding to each of them. Each response increments the IP ID. Some hosts use a unique IP ID sequence for each IP address they communicate with. If that had been the case, we would not have seen the IP ID leaping like this and we would have to look for a different target host on the network.

1. Repeat step 3 using spoofed addresses that you suspect may be allowed through the firewall or trusted. Try addresses behind their firewall, as well as the [RFC 1918](http://www.rfc-editor.org/rfc/rfc1918.txt) private networks such as 10.0.0.0/8, 192.168.0.0/16, and 172.16.0.0/12. Also try localhost (127.0.0.1) and maybe another address from 127.0.0.0/8 to detect cases where 127.0.0.1 is hard coded in. There have been many security holes related to spoofed localhost packets, including the infamous Land denial of service attack. Misconfigured systems sometimes trust these addresses without checking whether they came from the loopback interface. If a source address gets through to the end host, the IP ID will jump as seen in step 3. If it continues to increment slowly as in step 2, the packets were likely dropped by a firewall or router.

The end result of this technique is a list of source address netblocks that are permitted through the firewall, and those that are blocked. This information is valuable for several reasons. The IP addresses a company chooses to block or allow may give clues as to what addresses are used internally or trusted. For example, machines on a company's production network might trust IP addresses on the corporate network, or trust a system administrator's personal machine. Machines on the same production network also sometimes trust each other, or trust localhost. Common IP-based trust relationships are seen in NFS exports, host firewall rules, TCP wrappers, custom applications, rlogin, etc. Another example is SNMP, where a spoofed request to a Cisco router could cause the router to transfer (TFTP) its configuration data back to the attacker. Before spending substantial time trying to find and exploit these problems, use the test described here to determine whether the spoofed packets even get through.

A concrete example of this trusted-source-address problem is that I once found that a company's custom UDP service allowed users to skip authentication if they came from special netblocks entered into a configuration file. These netblocks corresponded to different corporate locations, and the feature was meant to ease administration and debugging. Their Internet-facing firewall smartly tried to block those addresses, as actual employees could access production from a private link instead. But by using the techniques described in this section, I found that the firewall was not perfectly synced with the config file. There were a few addresses from which I could successfully forge the UDP control messages and take over their application.

This technique of mapping out the firewall rules does not use Nmap, but the results are valuable for future runs. For example, this test can show whether to use certain decoys (-D). The best decoys will make it all the way to the target system. In addition, forged packets must get through for the IP ID idle scan (discussed later) to work. Testing potential source IPs with this technique is usually easier than finding and testing every potential idle proxy machine on a network. Potential idle proxies need only be tested if they pass step number two, above.

### UDP Version Scanning

The previous sections have all focused on the prevalent TCP protocol. Working with UDP is often more difficult because the protocol does not provide acknowledgment of open ports like TCP does. Many UDP applications will simply ignore unexpected packets, leaving Nmap unsure whether the port is open or filtered. So Nmap places these ambiguous ports in the open|filtered state, as shown in [Example 10.4](https://nmap.org/book/determining-firewall-rules.html#defeating-firewalls-udp-scan).

Example 10.4. UDP scan against firewalled host

# **nmap -sU -p50-59 scanme.nmap.org**

Starting Nmap ( https://nmap.org )

Nmap scan report for scanme.nmap.org (64.13.134.52)

PORT STATE SERVICE

50/udp open|filtered re-mail-ck

51/udp open|filtered la-maint

52/udp open|filtered xns-time

53/udp open|filtered domain

54/udp open|filtered xns-ch

55/udp open|filtered isi-gl

56/udp open|filtered xns-auth

57/udp open|filtered priv-term

58/udp open|filtered xns-mail

59/udp open|filtered priv-file

Nmap done: 1 IP address (1 host up) scanned in 1.38 seconds

This 10-port scan was not very helpful. No port responded to the probe packets, and so they are all listed as open or filtered. One way to better understand which ports are actually open is to send a whole bunch of UDP probes for dozens of different known UDP services in the hope of eliciting a response from any open ports. Nmap version detection ([Chapter 7, Service and Application Version Detection](https://nmap.org/book/vscan.html)) does exactly that. [Example 10.5](https://nmap.org/book/determining-firewall-rules.html#defeating-firewalls-udp-version-scan) shows the same scan with the addition of version detection (-sV).

Example 10.5. UDP version scan against firewalled host

# **nmap -sV -sU -p50-59 scanme.nmap.org**

Starting Nmap ( https://nmap.org )

Nmap scan report for scanme.nmap.org (64.13.134.52)

PORT STATE SERVICE VERSION

50/udp open|filtered re-mail-ck

51/udp open|filtered la-maint

52/udp open|filtered xns-time

53/udp open domain ISC BIND 9.3.4

54/udp open|filtered xns-ch

55/udp open|filtered isi-gl

56/udp open|filtered xns-auth

57/udp open|filtered priv-term

58/udp open|filtered xns-mail

59/udp open|filtered priv-file

Nmap done: 1 IP address (1 host up) scanned in 56.59 seconds

Version detection shows beyond a doubt that port 53 (domain) is open, and even what it is running. The other ports are still open|filtered because they did not respond to any of the probes. They are probably filtered, though this is not guaranteed. They could be running a service such as SNMP which only responds to packets with the correct community string. Or they could be running an obscure or custom UDP service for which no Nmap version detection probe exists. Also note that this scan took more than 40 times as long as the previous scan. Sending all of those probes to each port is a relatively slow process. Adding the --version-intensity 0 option would reduce scan time significantly by only sending the probes most likely to elicit a response from services at a given port number.

[http://172.30.2.67:8000](http://172.30.2.67:8000/)

[http://172.30.2.67:8001](http://172.30.2.67:8001/)

<http://172.30.2.67:8003>

5 :Firewall Evasion

Many Internet pioneers envisioned a global open network with a universal IP address space allowing virtual connections between any two nodes. This allows hosts to act as true peers, serving and retrieving information from each other. People could access all of their home systems from work, changing the climate control settings or unlocking the doors for early guests. This vision of universal connectivity has been stifled by address space shortages and security concerns. In the early 1990s, organizations began deploying firewalls for the express purpose of reducing connectivity. Huge networks were cordoned off from the unfiltered Internet by application proxies, network address translation, and packet filters. The unrestricted flow of information gave way to tight regulation of approved communication channels and the content that passes over them.

Network obstructions such as firewalls can make mapping a network exceedingly difficult. It will not get any easier, as stifling casual reconnaissance is often a key goal of implementing the devices. Nevertheless, Nmap offers many features to help understand these complex networks, and to verify that filters are working as intended. It even supports mechanisms for bypassing poorly implemented defenses. One of the best methods of understanding your network security posture is to try to defeat it. Place yourself in the mind-set of an attacker, and deploy techniques from this section against your networks. Launch an FTP bounce scan, idle scan, fragmentation attack, or try to tunnel through one of your own proxies.

In addition to restricting network activity, companies are increasingly monitoring traffic with intrusion detection systems (IDS). All of the major IDSs ship with rules designed to detect Nmap scans because scans are sometimes a precursor to attacks. Many of these products have recently morphed into intrusion prevention systems (IPS) that actively block traffic deemed malicious. Unfortunately for network administrators and IDS vendors, reliably detecting bad intentions by analyzing packet data is a tough problem. Attackers with patience, skill, and the help of certain Nmap options can usually pass by IDSs undetected. Meanwhile, administrators must cope with large numbers of false positive results where innocent activity is misdiagnosed and alerted on or blocked.

Occasionally people suggest that Nmap should not offer features for evading firewall rules or sneaking past IDSs. They argue that these features are just as likely to be misused by attackers as used by administrators to enhance security. The problem with this logic is that these methods would still be used by attackers, who would just find other tools or patch the functionality into Nmap. Meanwhile, administrators would find it that much harder to do their jobs. Deploying only modern, patched FTP servers is a far more powerful defense than trying to prevent the distribution of tools implementing the FTP bounce attack.

There is no magic bullet (or Nmap option) for detecting and subverting firewalls and IDS systems. It takes skill and experience. A tutorial is beyond the scope of this reference guide, which only lists the relevant options and describes what they do.

-f (fragment packets); --mtu (using the specified MTU)

The -f option causes the requested scan (including host discovery scans) to use tiny fragmented IP packets. The idea is to split up the TCP header over several packets to make it harder for packet filters, intrusion detection systems, and other annoyances to detect what you are doing. Be careful with this! Some programs have trouble handling these tiny packets. The old-school sniffer named Sniffit segmentation faulted immediately upon receiving the first fragment. Specify this option once, and Nmap splits the packets into eight bytes or less after the IP header. So a 20-byte TCP header would be split into three packets. Two with eight bytes of the TCP header, and one with the final four. Of course each fragment also has an IP header. Specify -f again to use 16 bytes per fragment (reducing the number of fragments). Or you can specify your own offset size with the --mtu option. Don't also specify -f if you use --mtu. The offset must be a multiple of eight. While fragmented packets won't get by packet filters and firewalls that queue all IP fragments, such as the CONFIG\_IP\_ALWAYS\_DEFRAG option in the Linux kernel, some networks can't afford the performance hit this causes and thus leave it disabled. Others can't enable this because fragments may take different routes into their networks. Some source systems defragment outgoing packets in the kernel. Linux with the iptables connection tracking module is one such example. Do a scan while a sniffer such as Wireshark is running to ensure that sent packets are fragmented. If your host OS is causing problems, try the --send-eth option to bypass the IP layer and send raw ethernet frames.

Fragmentation is only supported for Nmap's raw packet features, which includes TCP and UDP port scans (except connect scan and FTP bounce scan) and OS detection. Features such as version detection and the Nmap Scripting Engine generally don't support fragmentation because they rely on your host's TCP stack to communicate with target services.

-D *<decoy1>*[,*<decoy2>*][,ME][,...] (Cloak a scan with decoys)

Causes a decoy scan to be performed, which makes it appear to the remote host that the host(s) you specify as decoys are scanning the target network too. Thus their IDS might report 5–10 port scans from unique IP addresses, but they won't know which IP was scanning them and which were innocent decoys. While this can be defeated through router path tracing, response-dropping, and other active mechanisms, it is generally an effective technique for hiding your IP address.

Separate each decoy host with commas, and you can optionally use ME as one of the decoys to represent the position for your real IP address. If you put ME in the sixth position or later, some common port scan detectors (such as Solar Designer's excellent Scanlogd) are unlikely to show your IP address at all. If you don't use ME, Nmap will put you in a random position. You can also use RND to generate a random, non-reserved IP address, or RND:*<number>* to generate *<number>* addresses.

Note that the hosts you use as decoys should be up or you might accidentally SYN flood your targets. Also it will be pretty easy to determine which host is scanning if only one is actually up on the network. You might want to use IP addresses instead of names (so the decoy networks don't see you in their nameserver logs). Right now random IP address generation is only supported with IPv4

Decoys are used both in the initial host discovery scan (using ICMP, SYN, ACK, or whatever) and during the actual port scanning phase. Decoys are also used during remote OS detection (-O). Decoys do not work with version detection or TCP connect scan. When a scan delay is in effect, the delay is enforced between each batch of spoofed probes, not between each individual probe. Because decoys are sent as a batch all at once, they may temporarily violate congestion control limits.

It is worth noting that using too many decoys may slow your scan and potentially even make it less accurate. Also, some ISPs will filter out your spoofed packets, but many do not restrict spoofed IP packets at all.

-S *<IP\_Address>* (Spoof source address)

In some circumstances, Nmap may not be able to determine your source address (Nmap will tell you if this is the case). In this situation, use -S with the IP address of the interface you wish to send packets through.

Another possible use of this flag is to spoof the scan to make the targets think that someone else is scanning them. Imagine a company being repeatedly port scanned by a competitor! The -e option and -Pn are generally required for this sort of usage. Note that you usually won't receive reply packets back (they will be addressed to the IP you are spoofing), so Nmap won't produce useful reports.

-e *<interface>* (Use specified interface)

Tells Nmap what interface to send and receive packets on. Nmap should be able to detect this automatically, but it will tell you if it cannot.

--source-port *<portnumber>*; -g *<portnumber>* (Spoof source port number)

One surprisingly common misconfiguration is to trust traffic based only on the source port number. It is easy to understand how this comes about. An administrator will set up a shiny new firewall, only to be flooded with complaints from ungrateful users whose applications stopped working. In particular, DNS may be broken because the UDP DNS replies from external servers can no longer enter the network. FTP is another common example. In active FTP transfers, the remote server tries to establish a connection back to the client to transfer the requested file.

Secure solutions to these problems exist, often in the form of application-level proxies or protocol-parsing firewall modules. Unfortunately there are also easier, insecure solutions. Noting that DNS replies come from port 53 and active FTP from port 20, many administrators have fallen into the trap of simply allowing incoming traffic from those ports. They often assume that no attacker would notice and exploit such firewall holes. In other cases, administrators consider this a short-term stop-gap measure until they can implement a more secure solution. Then they forget the security upgrade.

Overworked network administrators are not the only ones to fall into this trap. Numerous products have shipped with these insecure rules. Even Microsoft has been guilty. The IPsec filters that shipped with Windows 2000 and Windows XP contain an implicit rule that allows all TCP or UDP traffic from port 88 (Kerberos). In another well-known case, versions of the Zone Alarm personal firewall up to 2.1.25 allowed any incoming UDP packets with the source port 53 (DNS) or 67 (DHCP).

Nmap offers the -g and --source-port options (they are equivalent) to exploit these weaknesses. Simply provide a port number and Nmap will send packets from that port where possible. Most scanning operations that use raw sockets, including SYN and UDP scans, support the option completely. The option notably doesn't have an effect for any operations that use normal operating system sockets, including DNS requests, TCP connect scan, version detection, and script scanning. Setting the source port also doesn't work for OS detection, because Nmap must use different port numbers for certain OS detection tests to work properly.

--data *<hex string>* (Append custom binary data to sent packets)

This option lets you include binary data as payload in sent packets. *<hex string>* may be specified in any of the following formats: 0xAABBCCDDEEFF*<...>*, AABBCCDDEEFF*<...>* or \xAA\xBB\xCC\xDD\xEE\xFF*<...>*. Examples of use are --data 0xdeadbeef and --data \xCA\xFE\x09. Note that if you specify a number like 0x00ff no byte-order conversion is performed. Make sure you specify the information in the byte order expected by the receiver.

--data-string *<string>* (Append custom string to sent packets)

This option lets you include a regular string as payload in sent packets. *<string>* can contain any string. However, note that some characters may depend on your system's locale and the receiver may not see the same information. Also, make sure you enclose the string in double quotes and escape any special characters from the shell. Examples: --data-string "Scan conducted by Security Ops, extension 7192" or --data-string "Ph34r my l33t skills". Keep in mind that nobody is likely to actually see any comments left by this option unless they are carefully monitoring the network with a sniffer or custom IDS rules.

--data-length *<number>* (Append random data to sent packets)

Normally Nmap sends minimalist packets containing only a header. So its TCP packets are generally 40 bytes and ICMP echo requests are just 28. Some UDP ports and IP protocols get a custom payload by default. This option tells Nmap to append the given number of random bytes to most of the packets it sends, and not to use any protocol-specific payloads. (Use --data-length 0 for no random or protocol-specific payloads. OS detection (-O) packets are not affected because accuracy there requires probe consistency, but most pinging and portscan packets support this. It slows things down a little, but can make a scan slightly less conspicuous.

--ip-options *<S|R [route]|L [route]|T|U ... >*; --ip-options *<hex string>* (Send packets with specified ip options)

The [IP protocol](http://www.rfc-editor.org/rfc/rfc791.txt) offers several options which may be placed in packet headers. Unlike the ubiquitous TCP options, IP options are rarely seen due to practicality and security concerns. In fact, many Internet routers block the most dangerous options such as source routing. Yet options can still be useful in some cases for determining and manipulating the network route to target machines. For example, you may be able to use the record route option to determine a path to a target even when more traditional traceroute-style approaches fail. Or if your packets are being dropped by a certain firewall, you may be able to specify a different route with the strict or loose source routing options.

The most powerful way to specify IP options is to simply pass in values as the argument to --ip-options. Precede each hex number with \x then the two digits. You may repeat certain characters by following them with an asterisk and then the number of times you wish them to repeat. For example, \x01\x07\x04\x00\*36\x01 is a hex string containing 36 NUL bytes.

Nmap also offers a shortcut mechanism for specifying options. Simply pass the letter R, T, or U to request record-route, record-timestamp, or both options together, respectively. Loose or strict source routing may be specified with an L or S followed by a space and then a space-separated list of IP addresses.

If you wish to see the options in packets sent and received, specify --packet-trace. For more information and examples of using IP options with Nmap, see <https://seclists.org/nmap-dev/2006/q3/52>.

--ttl *<value>* (Set IP time-to-live field)

Sets the IPv4 time-to-live field in sent packets to the given value.

--randomize-hosts (Randomize target host order)

Tells Nmap to shuffle each group of up to 16384 hosts before it scans them. This can make the scans less obvious to various network monitoring systems, especially when you combine it with slow timing options. If you want to randomize over larger group sizes, increase PING\_GROUP\_SZ in nmap.h and recompile. An alternative solution is to generate the target IP list with a list scan (-sL -n -oN *<filename>*), randomize it with a Perl script, then provide the whole list to Nmap with -iL.

--spoof-mac *<MAC address, prefix, or vendor name>* (Spoof MAC address)

Asks Nmap to use the given MAC address for all of the raw ethernet frames it sends. This option implies --send-eth to ensure that Nmap actually sends ethernet-level packets. The MAC given can take several formats. If it is simply the number 0, Nmap chooses a completely random MAC address for the session. If the given string is an even number of hex digits (with the pairs optionally separated by a colon), Nmap will use those as the MAC. If fewer than 12 hex digits are provided, Nmap fills in the remainder of the six bytes with random values. If the argument isn't a zero or hex string, Nmap looks through nmap-mac-prefixes to find a vendor name containing the given string (it is case insensitive). If a match is found, Nmap uses the vendor's OUI (three-byte prefix) and fills out the remaining three bytes randomly. Valid --spoof-mac argument examples are Apple, 0, 01:02:03:04:05:06, deadbeefcafe, 0020F2, and Cisco. This option only affects raw packet scans such as SYN scan or OS detection, not connection-oriented features such as version detection or the Nmap Scripting Engine.

--proxies *<Comma-separated list of proxy URLs>* (Relay TCP connections through a chain of proxies)

Asks Nmap to establish TCP connections with a final target through supplied chain of one or more HTTP or SOCKS4 proxies. Proxies can help hide the true source of a scan or evade certain firewall restrictions, but they can hamper scan performance by increasing latency. Users may need to adjust Nmap timeouts and other scan parameters accordingly. In particular, a lower --max-parallelism may help because some proxies refuse to handle as many concurrent connections as Nmap opens by default.

This option takes a list of proxies as argument, expressed as URLs in the format proto://host:port. Use commas to separate node URLs in a chain. No authentication is supported yet. Valid protocols are HTTP and SOCKS4.

Warning: this feature is still under development and has limitations. It is implemented within the nsock library and thus has no effect on the ping, port scanning and OS discovery phases of a scan. Only NSE and version scan benefit from this option so far—other features may disclose your true address. SSL connections are not yet supported, nor is proxy-side DNS resolution (hostnames are always resolved by Nmap).

--badsum (Send packets with bogus TCP/UDP checksums)

Asks Nmap to use an invalid TCP, UDP or SCTP checksum for packets sent to target hosts. Since virtually all host IP stacks properly drop these packets, any responses received are likely coming from a firewall or IDS that didn't bother to verify the checksum. For more details on this technique, see <https://nmap.org/p60-12.html>

--adler32 (Use deprecated Adler32 instead of CRC32C for SCTP checksums)

Asks Nmap to use the deprecated Adler32 algorithm for calculating the SCTP checksum. If --adler32 is not given, CRC-32C (Castagnoli) is used. [RFC 2960](http://www.rfc-editor.org/rfc/rfc2960.txt) originally defined Adler32 as checksum algorithm for SCTP; [RFC 4960](http://www.rfc-editor.org/rfc/rfc4960.txt) later redefined the SCTP checksums to use CRC-32C. Current SCTP implementations should be using CRC-32C, but in order to elicit responses from old, legacy SCTP implementations, it may be preferable to use Adler32.

[http://172.30.2.67:8000](http://172.30.2.67:8000/)

[http://172.30.2.67:8001](http://172.30.2.67:8001/)

<http://172.30.2.67:8003>