[Skip to main content](https://lms.alnafi.com/xblock/block-v1:alnafi+DCCS102+2025_DCCS+type@vertical+block@2f9e22ed1e3c45c48476a69bc7433e53?exam_access=&recheck_access=1&show_bookmark=0&show_title=0&view=student_view#main)

**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 mindset 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 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 originally defined Adler32 as checksum algorithm for SCTP; RFC 4960 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.