NAME

nmap − Network exploration tool and security scanner

SYNOPSIS

nmap [Scan Type(s)] [Options] <host or net #1 ... [#N]>

DESCRIPTION

Nmap is designed to allow system administrators and curious individuals to scan large networks to determine which hosts are up and what services they are offering. *nmap* supports a large number of scanning techniques such as: UDP, TCP connect(), TCP SYN (half open), ftp proxy (bounce attack), Reverse-ident, ICMP (ping sweep), FIN, ACK sweep, Xmas Tree, SYN sweep, and Null scan. See the *Scan Types* section for more details. nmap also offers a number of advanced features such as remote OS detection via TCP/IP fingerprinting, stealth scanning, dynamic delay and retransmission calculations, parallel scanning, detection of down hosts via parallel pings, decoy scanning, port filtering detection, direct (non-portmapper) RPC scanning, fragmentation scanning, and flexible target and port specification.

Significant effort has been put into decent nmap performance for non-root users. Unfortunately, many critical kernel interfaces (such as raw sockets) require root privileges. nmap should be run as root whenever possible.

The result of running nmap is usually a list of interesting ports on the machine(s) being scanned (if any). Nmap always gives the port's "well known" service name (if any), number, state, and protocol. The state is either 'open', ´filtered´, or ´unfiltered´. Open means that the target machine will accept() connections on that port. Filtered means that a firewall, filter, or other network obstacle is covering the port and preventing nmap from determining whether the port is open. Unfiltered means that the port is known by nmap to be closed and no firewall/filter seems to be interfering with nmap's attempts to determine this. Unfiltered ports are the common case and are only shown when most of the scanned ports are in the filtered state.

Depending on options used, nmap may also report the following characteristics of the remote host: OS in use, TCP sequencability, usernames running the programs which have bound to each port, the DNS name, whether the host is a smurf address, and a few others.

OPTIONS

Options that make sense together can generally be combined. Some options are specific to certain scan modes. *nmap* tries to catch and warn the user about psychotic or unsupported option combinations.

If you are impatient, you can skip to the *examples* section at the end, which demonstrates common usage. You can also run **nmap -h** for a quick reference page listing all the options.

SCAN TYPES

−sT TCP connect() scan: This is the most basic form of TCP scanning. The connect() system call provided by your operating system is used to open a connection to every interesting port on the machine. If the port is listening, connect() will succeed, otherwise the port isn't reachable. One strong advantage to this technique is that you don't need any special privileges. Any user on most UNIX boxes is free to use this call.

This sort of scan is easily detectable as target host logs will show a bunch of connection and error messages for the services which accept() the connection just to have it immediately shutdown.

−sS TCP SYN scan: This technique is often referred to as "half-open" scanning, because you don't open a full TCP connection. You send a SYN packet, as if you are going to open a real connection and you wait for a response. A SYN|ACK indicates the port is listening. A RST is indicative of a non−listener. If a SYN|ACK is received, a RST is immediately sent to tear down the connection (actually our OS kernel does this for us). The primary advantage to this scanning technique is that fewer sites will log it. Unfortunately you need root privileges to build these custom SYN packets.

−sF −sX −sN

Stealth FIN, Xmas Tree, or Null scan modes: There are times when even SYN scanning isn't clandestine enough. Some firewalls and packet filters watch for SYNs to restricted ports, and programs like Synlogger and Courtney are available to detect these scans. These advanced scans, on the other hand, may be able to pass through unmolested.

The idea is that closed ports are required to reply to your probe packet with an RST, while open ports must ignore the packets in question (see RFC 793 pp 64). The FIN scan uses a bare (surprise) FIN packet as the probe, while the Xmas tree scan turns on the FIN, URG, and PUSH flags. The Null scan turns off all flags. Unfortunately Microsoft (like usual) decided to completely ignore the standard and do things their own way. Thus this scan type will not work against systems running Windows95/NT. On the positive side, this is a good way to distinguish between the two platforms. If the scan finds open ports, you know the machine is not a Windows box. If a -sF,-sX,or -sN scan shows all ports closed, yet a SYN (-sS) scan shows ports being opened, you are probably looking at a Windows box. This is less useful now that nmap has proper OS detection built in. There are also a few other systems that are broken in the same way Windows is. They include Cisco, BSDI, HP/UX, MVS, and IRIX. All of the above send resets from the open ports when they should just drop the packet.

−sP Ping scanning: Sometimes you only want to know which hosts on a network are up. Nmap can do this by sending ICMP echo request packets to every IP address on the networks you specify. Hosts that respond are up. Unfortunately, some sites such as microsoft.com block echo request packets. Thus nmap can also send a TCP ack packet to (by default) port 80. If we get an RST back, that machine is up. A third technique involves sending a SYN packet and waiting for a RST or a SYN/ACK. For non-root users, a connect() method is used.

By default (for root users), nmap uses both the ICMP and ACK techniques in parallel. You can change the **−P** option described later.

Note that pinging is done by default anyway, and only hosts that respond are scanned. Only use this option if you wish to ping sweep **without** doing any actual port scans.

−sU UDP scans: This method is used to determine which UDP (User Datagram Protocol, RFC 768) ports are open on a host. The technique is to send 0 byte udp packets to each port on the target machine. If we receive an ICMP port unreachable message, then the port is closed. Otherwise we assume it is open.

Some people think UDP scanning is pointless. I usually remind them of the recent Solaris rcpbind hole. Rpcbind can be found hiding on an undocumented UDP port somewhere above 32770. So it doesn’t matter that 111 is blocked by the firewall. But can you find which of the more than 30,000 high ports it is listening on? With a UDP scanner you can! There is also the cDc Back Orifice backdoor program which hides on a configurable UDP port on Windows machines. Not to mention the many commonly vulnerable services that utilize UDP such as snmp, tftp, NFS, etc.

Unfortunately UDP scanning is sometimes painfully slow since most hosts impliment a suggestion in RFC 1812 (section 4.3.2.8) of limiting the ICMP error message rate. For example, the Linux kernel (in net/ipv4/icmp.h) limits destination unreachable message generation to 80 per 4 seconds, with a 1/4 second penalty if that is exceeded. Solaris has much more strict limits (about 2 messages per second) and thus takes even longer to scan. *nmap* detects this rate limiting and slows down accordingly, rather than flood the network with useless packets that will be ignored by the target machine.

As is typical, Microsoft ignored the suggestion of the RFC and does not seem to do any rate limiting at all on Win95 and NT machines. Thus we can scan all 65K ports of a Windows machine **very** quickly. Woop!

−sA ACK scan: This advanced method is usually used to map out firewall rulesets. In particular, it can help determine whether a firewall is stateful or just a simple packet filter that blocks incoming SYN packets.

This scan type sends an ACK packet (with random looking acknowledgement/sequence numbers) to the ports specified. If a RST comes back, the ports is classified as "unfiltered". If nothing comes back (or if an ICMP unreachable is returned), the port is classified as "filtered". Note that *nmap* usually doesn’t print "unfiltered" ports, so getting **no** ports shown in the output is usually a sign that all the probes got through (and returned RSTs). This scan will obviously never show ports in the "open" state.

−sW Window scan: This advanced scan is very similar to the ACK scan, except that it can sometimes detect open ports as well as filtered/nonfiltered due to an anomaly in the TCP window size reporting by some operating systems. Systems vulnerable to this include at least some versions of AIX, Amiga, BeOS, BSDI, Cray, Tru64 UNIX, DG/UX, OpenVMS, Digital UNIX, FreeBSD, HP-UX, OS/2, IRIX, MacOS, NetBSD, OpenBSD, OpenStep, QNX, Rhapsody, SunOS 4.X, Ultrix, VAX, and VxWorks. See the nmap-hackers mailing list archive for a full list.

−sR RPC scan. This method works in combination with the various port scan methods of Nmap. It takes all the TCP/UDP ports found open and then floods them with SunRPC program NULL commands in an attempt to determine whether they are RPC ports, and if so, what program and version number they serve up. Thus you can effectively obtain the same info as firewall (or protected by TCP wrappers). Decoys do not currently work with RPC scan, at some point I may add decoy support for UDP RPC scans.

−b <ftp relay host>
FTP bounce attack: An interesting "feature" of the ftp protocol (RFC 959) is support for "proxy" ftp connections. In other words, I should be able to connect from evil.com to the FTP server of target.com and request that the server send a file ANYWHERE on the internet! Now this may have worked well in 1985 when the RFC was written. But in today's Internet, we can't have people hijacking ftp servers and requesting that data be spit out to arbitrary points on the internet. As *Hobbit* wrote back in 1995, this protocol flaw "can be used to post virtually untraceable mail and news, hammer on servers at various sites, fill up disks, try to hop firewalls, and generally be annoying and hard to track down at the same time." What we will exploit this for is to (surprise, surprise) scan TCP ports from a "proxy" ftp server. Thus you could connect to an ftp server behind a firewall, and then scan ports that are more likely to be blocked (139 is a good one). If the ftp server allows reading from and writing to some directory (such as /incoming), you can send arbitrary data to ports that you do find open (nmap doesn't do this for you though).

The argument passed to the 'b' option is the host you want to use as a proxy, in standard URL notation. The format is: *username:password@server:port.* Everything but *server* is optional. To determine what servers are vulnerable to this attack, you can see my article in *Phrack* 51. And updated version is available at the *nmap* URL (http://www.insecure.org/nmap).

GENERAL OPTIONS
None of these are required but some can be quite useful.

−P0 Do not try and ping hosts at all before scanning them. This allows the scanning of networks that don't allow ICMP echo requests (or responses) through their firewall. microsoft.com is an example of such a network, and thus you should always use **−P0** or **−PT80** when portscanning microsoft.com.

−PT Use TCP "ping" to determine what hosts are up. Instead of sending ICMP echo request packets and waiting for a response, we spew out TCP ACK packets throughout the target network (or to a single machine) and then wait for responses to trickle back. Hosts that are up should respond with a RST. This option preserves the efficiency of only scanning hosts that are up while still allowing you to scan networks/hosts that block ping packets. For non root users, we use connect(). To set the destination port of the probe packets use -PT<port number>. The default port is 80, since this port is often not filtered out.

−PS This option uses SYN (connection request) packets instead of ACK packets for root users. Hosts that are up should respond with a RST (or, rarely, a SYN|ACK).

−PI This option uses a true ping (ICMP echo request) packet. It finds hosts that are up and also looks for subnet-directed broadcast addresses on your network. These are IP addresses which are externally reachable and translate to a broadcast of incomming IP packets to a subnet of computers. These should be eliminated if found as they allow for numerous denial of service attacks (Smurf is the most common).

−PB This is the default ping type. It uses both the ACK (**−PT**) and ICMP (**−PI**) sweeps in parallel. This way you can get firewalls that filter either one (but not both).

−O This option activates remote host identification via TCP/IP fingerprinting. In other words, it uses a bunch of techniques to detect subtleties in the underlying operating system network stack of the computers you are scanning. It uses this information to create a 'fingerprint' which it compares with its database of known OS fingerprints (the nmap-os-fingerprints file) to decide what type of system you are scanning.

If you find a machine that is misdiagnosed and has at least one port open, it would be useful if you mail me the details (ie OS blah version foo was detected as OS blah version bar). If you find a machine with at least one port open for which nmap says 'unknown operating system', then it would be useful if you send me the IP address along with the OS name and version number. If you can't send the IP address, the next best thing is to run nmap with the **−d** option and send me the three fingerprints that should result along with the OS name and version number. By doing this you contribute to the pool of operating systems known to nmap and thus it will be more accurate for everyone.

−I This turns on TCP reverse ident scanning. As noted by Dave Goldsmith in a 1996 Bugtraq post, the ident protocol (rfc 1413) allows for the disclosure of the username that owns any process connected via TCP, even if that process didn't initiate the connection. So you can, for example, connect to the http port and then use identd to find out whether the server is running as root. This can only be done with a full TCP connection to the target port (i.e. the -sT scanning option). When **−I** is used, the remote host's identd is queried for each open port found. Obviously this won't work if the host is not running identd.

−f This option causes the requested SYN, FIN, XMAS, or NULL scan 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. My favorite sniffer segmentation faulted immediately upon receiving the first 36-byte fragment. After that comes a 24 byte one! While this method won't get by packet filters and firewalls that queue all IP fragments (like 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.

Note that I do not yet have this option working on all systems. It works fine for my Linux, FreeBSD, and OpenBSD boxes and some people have reported success with other *NIX variants.

−v Verbose mode. This is a highly recommended option and it gives out more information about what is going on. You can use it twice for greater effect. Use **−d** a couple of times if you really want to get crazy with scrolling the screen!

−h This handy option display a quick reference screen of nmap usage options. As you may have noticed, this man page is not exactly a 'quick reference' :)

−oN <logfilename>
This logs the results of your scans in a normal **human readable** form into the file you specify as an argument.

−oM <logfilename>
This logs the results of your scans in a **machine parseable** form into the file you specify as an argument. You can give the argument ´-´ (without quotes) to shoot output into stdout (for shell pipelines, etc). In this case normal output will be suppressed. Watch out for error messages if you use this (they will still go to stderr). Also note that ´-v´ will cause some extra information to be printed.

−oS <logfilename>
thIs l0gz th3 r3suLtS of YouR ScanZ iN a **s|<ipT kiDd|3** f0rM iNto THe fiL3 U sPec fy 4s an arGuMEnT! U kAn gIv3 the 4rgument ´-´ (wItHOUt qUOteZ) to sh00t output iNT0 stDouT!@!!

−-resume <logfilename>
A network scan that is cancelled due to control-C, network outage, etc. can be resumed using this option. The logfilename must be either a normal (-oN) or machine parsable (-oM) log from the

aborted scan. No other options can be given (they will be the same as the aborted scan). Nmap will start on the machine after the last one successfully scanned in the log file.

−iL <inputfilename>
Reads target specifications from the file specified RATHER than from the command line. The file should contain a list of host or network expressions seperated by spaces, tabs, or newlines. Use a hyphen (-) as *inputfilename* if you want nmap to read host expressions from stdin (like at the end of a pipe). See the section *target specification* for more information on the expressions you fill the file with.

−iR This option tells Nmap to generate its own hosts to scan by simply picking random numbers :). It will never end. This can be useful for statistical sampling of the Internet to estimate various things. If you are ever really bored, try *nmap −sS −iR −p 80* to find some web servers to look at.

−p <port ranges>
This option specifies what ports you want to specify. For example '-p 23' will only try port 23 of the target host(s). ´−p 20-30,139,60000-´ scans ports between 20 and 30, port 139, and all ports greater than 60000. The default is to scan all ports between 1 and 1024 as well as any ports listed in the services file which comes with nmap.

−F Fast scan mode.
Specifies that you only wish to scan for ports listed in the services file which comes with nmap. This is obviously much faster than scanning all 65535 ports on a host.

−D <decoy1 [,decoy2][,ME],...>
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 extremely 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 you want your IP address to be used. If your put 'ME' in the 6th position or later, some common port scan detectors (such as Solar Designer's excellent scanlogd) are unlikeley to show your IP address at all. If you don't use 'ME', nmap will put you in a random position.

Note that the hosts you use as decoys should be up or you might accidently 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).

Also note that some (stupid) "port scan detectors" will firewall/deny routing to hosts that attempt port scans. Thus you might inadvertantly cause the machine you scan to lose connectivity with the decoy machines you are using. This could cause the target machines major problems if the decoy is, say, its internet gateway or even "localhost". Thus you might want to be careful of this option. The real moral of the story is that detectors of spoofable port scans should not take action against the machine that seems like it is port scanning them. It could just be a decoy!

Decoys are used both in the initial ping scan (using ICMP, SYN, ACK, or whatever) and during the actual port scanning phase. Decoys are also used during remote OS detection (**−O**).

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, although many (currently most) do not restrict spoofed IP packets at all.

−S <IP_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 your 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! This is not a supported usage (or the main purpose) of this flag. I just think it raises an interesting possibility that people should be aware of before they go accusing others of port scanning them. **−e** would generally be required for this sort of usage.

−e <interface>
Tells nmap what interface to send and receive packets on. Nmap should be able to detect this but it will tell you if it cannot.

−g <portnumber>
Sets the source port number used in scans. Many naive firewall and packet filter installations make an exception in their ruleset to allow DNS (53) or FTP-DATA (20) packets to come through and establish a connection. Obviously this completely subverts the security advantages of the firewall since intruders can just masquerade as FTP or DNS by modifying their source port. Obviously for a UDP scan you should try 53 first and TCP scans should try 20 before 53. Note that this is only a request -- nmap will honor it only if and when it is able to. For example, you can't do TCP ISN sampling all from one host:port to one host:port, so nmap changes the source port even if you used -g.

Be aware that there is a small performance penalty on some scans for using this option, because I sometimes store useful information in the source port number.

−r Tells Nmap **NOT** to randomize the order in which ports are scanned.

−−randomize_hosts
Tells Nmap to shuffle each group of up to 2048 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 (see below).

−M <max sockets>
Sets the maximum number of sockets that will be used in parallel for a TCP connect() scan (the default). This is useful to slow down the scan a little bit and avoid crashing remote machines. Another approach is to use −sS, which is generally easier for machines to handle.

TIMING OPTIONS
Generally Nmap does a good job at adjusting for Network characteristics at runtime and scanning as fast as possible while minimizing that chances of hosts/ports going undetected. However, there are same cases where Nmap's default timing policy may not meet your objectives. The following options provide a fine level of control over the scan timing:

-T <Paranoid|Sneaky|Polite|Normal|Aggressive|Insane>
These are canned timing policies for conveniently expressing your priorities to Nmap. **Paranoid** mode scans **very** slowly in the hopes of avoiding detection by IDS systems. It serializes all scans (no parallel scanning) and generally waits at least 5 minutes between sending packets. **Sneaky** is similar, except it only waits 15 seconds between sending packets. **Polite** is meant to ease load on the network and reduce the chances of crashing machines. It serializes the probes and waits **at least** 0.4 seconds between them. **Normal** is the default Nmap behaviour, which tries to run as quickly as possible without overloading the network or missing hosts/ports. **Aggressive** mode adds a 5 minute timeout per host and it never waits more than 1.25 seconds for probe responses. **Insane** is only suitable for very fast networks or where you don't mind losing some information. It times out hosts in 75 seconds and only waits 0.3 seconds for individual probes. It does allow for very quick network sweeps though :). You can also reference these by number (0-5). For example, ´-T 0´ gives you Paranoid mode and ´-T 5´ is Insane mode.

These canned timing modes should NOT be used in combination with the lower level controls given below.

--host_timeout <milliseconds>
Specifies the amount of time Nmap is allowed to spend scanning a single host before giving up on that IP. The default timing mode has no host timeout.

--max_rtt_timeout <milliseconds>
Specifies the maximum amount of time Nmap is allowed to wait for a probe response before retransmitting or timing out that particular probe. The default mode sets this to about 9000.

--min_rtt_timeout <milliseconds>
When the target hosts start to establish a pattern of responding very quickly, Nmap will shrink the amount of time given per probe. This speeds up the scan, but can lead to missed packets when a response takes longer than usual. With this parameter you can guarantee that Nmap will wait at least the given amount of time before giving up on a probe.

--initial_rtt_timeout <milliseconds>
Specifies the initial probe timeout. This is generally only useful when scanning firwalled hosts with -P0. Normally Nmap can obtain good RTT estimates from the ping and the first few probes. The default mode uses 6000.

--max_parallelism <number>
Specifies the maximum number of scans Nmap is allowed to perform in parallel. Setting this to one means Nmap will never try to scan more than 1 port at a time. It also effects other parallel scans such as ping sweep, RPC scan, etc.

--scan_delay <milliseconds>
Specifies the **minimum** amount of time Nmap must wait between probes. This is mostly useful to reduce network load or to slow the scan way down to sneak under IDS thresholds.

TARGET SPECIFICATION

Everything that isn't an option (or option argument) in nmap is treated as a target host specification. The simplest case is listing single hostnames or IP addresses on the command line. If you want to scan a subnet of IP addresses, you can append **'/mask'** to the hostname or IP address. **mask** must be between 0 (scan the whole internet) and 32 (scan the single host specified). Use /24 to scan a class 'C' address and /16 for a class 'B'.

Nmap also has a more powerful notation which lets you specify an IP address using lists/ranges for each element. Thus you can scan the whole class 'B' network 128.210.*.* by specifying '128.210.*.*' or '128.210.0-255.0-255' or even use the mask notation: '128.210.0.0/16'. These are all equivalent. If you use asterisks ('*'), remember that most shells require you to escape them with back slashes or protect them with quotes.

Another interesting thing to do is slice the Internet the other way. Instead of scanning all the hosts in a class specifying hosts to scan, see the *examples* section.

EXAMPLES

Here are some examples of using nmap, from simple and normal to a little more complex/esoteric. Note that actual numbers and some actual domain names are used to make things more concrete. In their place you should substitute addresses/names from **your own network.** I do not think portscanning other networks is illegal; nor should portscans be construed by others as an attack. I have scanned hundreds of thousands of machines and have received only one complaint. But I am not a lawyer and some (anal) people may be annoyed by *nmap* probes. Get permission first or use at your own risk.

nmap -v target.example.com

This option scans all reserved TCP ports on the machine target.example.com . The −v means turn on verbose mode.

nmap -sS -O target.example.com/24

Launches a stealth SYN scan against each machine that is up out of the 255 machines on class 'C' where target.example.com resides. It also tries to determine what operating system is running on each host that is up and running. This requires root privileges because of the SYN scan and the OS detection.

nmap -sX -p 22,53,110,143,4564 128.210.*.1-127

Sends an Xmas tree scan to the first half of each of the 255 possible 8 bit subnets in the 128.210 class 'B' address space. We are testing whether the systems run sshd, DNS, pop3d, imapd, or port 4564. Note that Xmas scan doesn't work on Microsoft boxes due to their deficient TCP stack. Same goes with CISCO, IRIX, HP/UX, and BSDI boxes.

nmap -v --randomize_hosts -p 80 '*.*.2.3-5'

Rather than focus on a specific IP range, it is sometimes interesting to slice up the entire Internet and scan a small sample from each slice. This command finds all web servers on machines with IP addresses ending in .2.3, .2.4, or .2.5 find more interesting machines starting at 127. so you might want to use '127-222' instead of the first asterisks because that section has a greater density of interesting machines (IMHO).

host -l company.com | cut '-d ' -f 4 | ./nmap -v -iL -

Do a DNS zone transfer to find the hosts in company.com and then feed the IP addresses to *nmap*. The above commands are for my GNU/Linux box. You may need different commands/options on other operating systems.

BUGS

Bugs? What bugs? Send me any that you find. Patches are nice too :) Remember to also send in new OS fingerprints so we can grow the database. Nmap will give you a submission URL when an appropriate fingerprint is found.

AUTHOR

Fyodor <*fyodor@dhp.com*>

DISTRIBUTION

The newest version of *nmap* can be obtained from *http://www.insecure.org/nmap/*

Source is provided to this software because we believe users have a right to know exactly what a program is going to do before they run it. This also allows you to audit the software for security holes (none have been found so far).

Source code also allows you to port nmap to new platforms, fix bugs, and add new features. You are highly encouraged to send your changes to Fyodor for possible incorporation into the main Nmap distribution. By sending these changes to Fyodor or nmap-hackers, it is assumed that you are offering Fyodor the unlimited, non-exclusive right to reuse, modify, and relicense the code. If you wish to specify special license conditions of your contributions, please state them up front.

It should also be noted that Nmap has been known to crash certain poorly written applications, TCP/IP stacks, and even operating systems. **Nmap should never be run against mission critical systems** unless you are prepared to suffer downtime. We acknowledge here that Nmap may crash your systems or networks and we disclaim all liability for any damage or problems Nmap could cause.

All versions of Nmap equal to or greater than 2.0 are believed to be Year 2000 (Y2K) compliant in all respects. There is no reason to believe versions earlier than 2.0 are susceptible to problems, but we have not tested them.