An Introduction to NMAP

Tim Corcoran

GSEC Practical Assignment
Version 1.2f
August 13, 2001

AN INTRODUCTION TO NMAP
Tim Corcoran
October 25, 2001

WHAT IS NMAP

NMAP is a multifaceted utility used to scan a range of IP addresses, identify active systems, determine which ports on those systems are open, and identify the respective operating systems. Like all security tools it can be used defensively, by a network manager, to identify weaknesses that need to be corrected, or offensively, by an attacker, probing for vulnerabilities to exploit.

In plain English, nmap will scan a range of host addresses or a network address range entered at the command line. It will determine which addresses are active systems currently on line. It will probe a range of ports, selectable by the user, to see what services the identified system is running. Finally it will probe the system for responses to some unusual packets to try and guess what operating system is installed on the target system.

The attacker who runs a careful and successful series of nmap scans on your network will know what systems are active and what exploits he or she should try to use to compromise the target system.

NMAP is free software offered under the terms of the GNU GPL. NMAP is downloadable, with its source code, from many sites on the Internet. It was originally written to run on Linux but is now available for several platforms.

The nmap program was written by Fyodor of insecure.org. He describes himself as a hacker, which he defines as being someone who enjoys “playing with computers, and pushing hardware and software to its limits.”[1] Fyodor wrote nmap because he was looking for a tool to simplify his exploration of the Internet. In his own words,

> Prior to writing nmap, I spent a lot of time with other scanners exploring the Internet and various private networks (note the avoidance of the "intranet" buzzword). I have used many of the top scanners available today, including strobe by Julian Assange, netcat by *Hobbit*, stcp by Uriel Maimon, pscan by Pluvius, ident-scan by Dave Goldsmith, and the SATAN tcp/udp scanners by Wietse Venema. These are all excellent scanners! In fact, I ended up hacking most of them to support the best features of the others. Finally I decided to write a whole new scanner, rather than rely on hacked versions of a dozen different scanners in my /usr/local/sbin. While I wrote all the code, nmap uses a lot of good ideas from its predecessors. I also incorporated some new stuff like

> fragmentation scanning and options that were on my "wish list" for other scanners. [1]

INTRODUCTION TO PORTS AND PORTSCANNING

A port is an access point for a service or application running on a computer system. All Internet and TCP/ip network based communications require a source IP address and source port as well as a destination IP address and destination port.

In his TCP/IP FAQ James compares IP addresses to street addresses for regular mail and port numbers to the unique individual at a street address who should get the delivery.

> I started off with an analogy to sending regular mail through the postal service. Sticking with that analogy, how do we ensure that they right person reads the mail we send? That mail may need to go to a particular person or department. With mail, we don't just include the P.O.Box number or street address of our destination - instead, we include the name of the person or department that should open the mail.
>
> Computers need something similar to make sure the correct software application on the destination computer gets the data packets we are sending, and some way to make sure replies get routed to the correct application on the computer at our end of the conversation. This is accomplished through the use of "port numbers".
>
> If you followed along with the discussion on IP routing with addresses, port numbers will be very easy to understand. We talked about how "IP" is a networking protocol and requires addresses. On top of that networking protocol, we make use of "transport" protocols to direct packets to specific software applications. The most common transport protocol on the Internet is Transport Control Protocol (or TCP). We also sometimes see something called the User Datagram Protocol (UDP) at the transport layer. Both are transport protocols, and both use port numbers (we'll talk about the differences between the two a little later on).
>
> Basically, just as with the need for a source and destination IP address in every packet, we must also include a source and destination port number in every packet. There are two types of port numbers to consider - port numbers used by server software, and port numbers used by client sofware.[2]

The Internet Assigned Numbers Authority assigns port numbers. It recognizes three categories of ports: the Well Known Ports, the Registered Ports, and the Dynamic and/or Private Ports. The Well Known Ports are those from 0 through 1023, the Registered Ports are those from 1024 through 49151, and the Dynamic or Private Ports are those from 49152 through 65535.

What are referred to as the well known ports (1-1023) are what the nmap program will scan if a port range is not specified on the command line.

A listing of all registered port numbers can be found at their web site at http://www.iana.org/assignments/port-numbers.

Jumping ahead of ourselves a little bit lets look at the partial output from an nmap scan to help illustrate what we are talking about. Our scanner sent packets to a system on the internet identified by its internet address. It further specified the delivery of each packet to a specific application or service on that system. Our scanner sent a packet to port 25 and got a response indicating that a service was in fact active at port 25. The well known ports list tells us that port 25 is the port reserved for SMTP, the simple mail transport protocol.

```
Port       State      Service
25/tcp     open       smtp
111/tcp    open       sunrpc
113/tcp    open       auth
515/tcp    open       printer
587/tcp    open       submission
1024/tcp   open       kdm
6000/tcp   open       X11
```

Port Scanning is the use of tools and utilities like nmap to automate the identification of active ports on a target system. These utilities use the target systems response to various types and variations of TCP/ip packets to identify these active ports.

INTRODUCTION TO OS FINGERPRINTING

Another useful feature of nmap is its ability to identify the operating system in use on the target system. This is important to an attacker because it will help them formulate a plan of attack with the highest possible probability of success. An attacker would be wasting his time trying a known Unix exploit against a WindowsNT host. Not only would it be unlikely to succeed but also the wasted activity increases the likelihood his activities will be noticed and traced.

According to Rik Farrow,

> When someone with half a clue decides to attack your system, he or she will first try to identify the operating system. Not every attack proceeds this way: Script kiddies probe huge address spaces looking for any system with a particular port open, which indicates that just maybe that system will be vulnerable. But for the professional penetration tester or hacker, identifying the operating system is an essential step in probing.[3]

He further states,

> the easiest way to identify operating systems is to run nmap. Nmap started off as a very functional network and port scanner, but in 1998 Fyodor added operating system fingerprinting techniques, and nmap has grown into the most feature-rich, free, stack-scanning tool in existence. Like many tools available on the Internet, this one is just as useful whether you are wearing a white hat or a black hat. Looking in the fingerprints file of a recent version of nmap (2.53), I counted fingerprints for 465 different stacks, including operating systems, routers, printers, and other networked devices.[3]

Nmap attempts to identify an operating system by probing the response of its TCP/ip stack. In their book HACKING EXPOSED, McClure,Scambray, and Kurtz explain stack fingerprinting as follows,

> …. it is important to explain exactly what stack fingerprinting is. Stack fingerprinting is an extremely powerful technology that allows you to quickly ascertain each host's operating system with a high degree of probability. Essentially, there are many nuances between one vendor's IP stack implementation versus another's. Vendors often interpret specific RFC guidance differently when writing their TCP / IP stack. Thus, by probing for these differences, we can begin to make an educated guess as to the exact operating system in use. For maximum reliability, stack fingerprinting generally requires at least one listening port. Nmap will make an educated guess about the operating system in use if no ports are open; however, the accuracy of such a guess will be fairly low. The definitive paper on the subject was written by Fyodor, first published in Phrack Magazine, and can be found at http://www.insecure.org/nmap/nmap-fingerprinting-article.html.[4]

Once the operating system of the target computer is identified the attacker can begin a concentrated effort to compromise it. This effort will most likely include the use of known exploits for the particular operating system. Make every effort to keep your systems at the most current OS levels and apply patches and security fixes as the OS vendor releases them.

At the current time there are over 500 OS fingerprints in the nmap-os-fingerprint file. Fyodor actively encourages people to submit to him (or to the website) any fingerprint they come across that will add to the database.

OBTAINING NMAP

NMAP can be obtained from a number of software archives on the web. Since it makes sense to go directly to the source I would recommend going to www.insecure.org, home of Fyodor, the author of NMAP. The site contains a great deal of information about the program. Including some basic installation instructions, documentation, information on mailing lists of interest to nmap users, and links to other sources of information.

INSTALLING NMAP ON LINUX

I installed nmap on an Intel Pentium II based laptop running Redhat Linux 7.0. I downloaded the files nmap-2.54BETA30-1.i386.rpm and nmap-frontend-0.2.54BETA30-1.i386.rpm. I ran the GnomeRPM utility provided with the OS to do the install, first on nmap and then on the front end.

The process is as follows:

Open GnomeRPM (go to the toolbar, choose system, then choose GnomeRPM)
Click install
Click add
Click on the nmap rpm in the directory you downloaded it to
Click install (again)

It is as simple as that.

INSTALLING NMAP ON A NON INTEL PLATFORM

In order to illustrate the installation and use of nmap on a non Intel platform I compiled and installed it on a Sun UltraSparc workstation, running Redhat Linux for Sparc, version 6.2.

The procedure for this installation follows:

- download the file nmap-2.54BETA30.tgz from the insecure.org website
- gunzip nmap-2.54BETA30.tgz
- tar –xvf nmap-2.54BETA30.tar
- cd nmap-2.54BETA30
- ./configure
- make
- make install

This tarball (nmap-2.54BETA30.tgz) also includes nmapfe.

NMAP FEATURES

Nmap is a very flexible and feature rich utility. Its operation can be very finely controlled through the mix of parameters entered at the command line when it is executed. It comes with a very informative and useful man page. It also provides a good help page when the –h option is used (see below):

nmap -h

```
Nmap V. 2.54BETA30 Usage: nmap [Scan Type(s)] [Options] <host or
net list>
Some Common Scan Types ('*' options require root privileges)
  -sT TCP connect() port scan (default)
```

```
* -sS TCP SYN stealth port scan (best all-around TCP scan)
* -sU UDP port scan
  -sP ping scan (Find any reachable machines)
* -sF,-sX,-sN Stealth FIN, Xmas, or Null scan (experts only)
  -sR/-I RPC/Identd scan (use with other scan types)
Some Common Options (none are required, most can be combined):
* -O Use TCP/IP fingerprinting to guess remote operating system
  -p <range> ports to scan.  Example range: '1-
1024,1080,6666,31337'
  -F Only scans ports listed in nmap-services
  -v Verbose. Its use is recommended.  Use twice for greater
effect.
  -P0 Don't ping hosts (needed to scan www.microsoft.com and
others)
* -Ddecoy_host1,decoy2[,...] Hide scan using many decoys
  -T <Paranoid|Sneaky|Polite|Normal|Aggressive|Insane> General
timing policy
  -n/-R Never do DNS resolution/Always resolve [default:
sometimes resolve]
  -oN/-oX/-oG <logfile> Output normal/XML/grepable scan logs to
<logfile>
  -iL <inputfile> Get targets from file; Use '-' for stdin
* -S <your_IP>/-e <devicename> Specify source address or network
interface
  --interactive Go into interactive mode (then press h for help)
Example: nmap -v -sS -O www.my.com 192.168.0.0/16 '192.88-90.*.*'
SEE THE MAN PAGE FOR MANY MORE OPTIONS, DESCRIPTIONS, AND
EXAMPLES
```

Nmap supports a variety of scan types. I will address some of them briefly here, and some in greater detail in the example packet capture sections. For a fuller description of the various supported scan types I suggest reading the man page or one of the resources listed at the end of this paper.

The most basic scan is the TCP connection scan. This is the scan that will be run if you do not enter a scan type parameter when executing nmap. It relies on the traditional TCP three way handshake. This scan is easily detected on the target machine and should be used sparingly. It is executed in nmap with *nmap –sT targetIPaddress*.

nmap -sS targetIPaddress will execute a SYN scan. This scan is also referred to as the "half open" scan. It executes the first two steps of the three way handshake then breaks the connection. It is covered in greater detail in the packet capture section of this paper.

The general format of the nmap command line is

nmap –parameters –options target_specification

for example

nmap –sS –v –O 192.168.1.106

This will execute nmap using a SYN scan (the –sS) with verbose output (the –v) and operating system identification turned on (the –O). It will be executed against the ip address 192.168.1.106.

A useful feature is the ability to log the output of nmap to a text file. The parameter –oN logfilename will save the output of the command to a plain old, human readable, text file. –oX logfilenamewill yield XML, -oG logfilename will yield machine grepable output, and (fyodor having a little fun with us) the –oS option which logs the output in script kiddie text.

An example of nmap with text file logging is:

nmap –sS –v –O –oN scanlog.txt 192.168.1.106

The –PI option will scan using true ICMP pings.
The –PO option will turn off pinging before a scan. Normally nmap will start a scan with an ICMP ping and then a TCP ping to each host designated as a target. If the hosts do not respond nmap assumes they are not online and does not do any further scanning to that address. This option would be used if you were scanning into a network that was screening ICMP.
The –O option activates OS identification.
The –I turns on reverse ident scanning which tries to determine the username of the owner of the process running on an open port. It relies on the ident protocol which is described in RFC1413. The goal here is to find exploitable services owned by root. Below is a sample of the output of a scan with the –I option set. Note the additional column in the output for the owner of the process.

nmap –v –sT –I 192.168.1.106

```
Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
Host  (192.168.1.106) appears to be up ... good.
Initiating Connect() Scan against  (192.168.1.106)
Adding open port 6000/tcp (owner: root)
Adding open port 113/tcp (owner: nobody)
Adding open port 587/tcp (owner: root)
Adding open port 515/tcp (owner: lp)
Adding open port 1024/tcp (owner: rpcuser)
Adding open port 25/tcp (owner: root)
Adding open port 111/tcp (owner: rpc)
The Connect() Scan took 1 second to scan 1549 ports.
Interesting ports on  (192.168.1.106):
(The 1542 ports scanned but not shown below are in state: closed)
Port       State       Service                 Owner
25/tcp     open        smtp                    root
111/tcp    open        sunrpc                  rpc
113/tcp    open        auth                    nobody
515/tcp    open        printer                 lp
587/tcp    open        submission              root
1024/tcp   open        kdm                     rpcuser
```

```
6000/tcp   open        X11                    root

Nmap run completed -- 1 IP address (1 host up) scanned in 82
seconds
```

The –p option allows you to specify a port or range of ports to scan. By default nmap will scan ports 1-1024 and any additional ports listed in the nmap services file.
The –iR option will tell nmap to just pick random host addresses to scan.
The –D option allows you to provide a list of host addresses that nmap will generate bogus traffic from. Every packet generated by your host during the scan will be replicated with each of the decoy ip addresses in the source address filed. If you provide a half a dozen decoy addresses it will appear to the target as if he is being scanned by seven hosts (yours plus the six decoys). nmap allows you to insert your own address where ever in the scanning order that you choose. According to the man page some scan detectors won't log addresses beyond the first six or so. Even if your address is logged it will not be immediately apparent to the victim who the real attacker is. One important note – the decoy IP addresses must be the addresses of live hosts.
The –T parameter allows you to adjust the timing of the scan. Scanning very slowly over a period of time is more likely to evade detection than a sudden blast of packets at every possible port on a system or host on a subnet. The possible choices are paranoid, sneaky, polite, normal, aggressive, and insane.

Nmap can be run in two ways. The first is from the command line, as we have discussed above, the second is with the assistance of a graphical front end. Josh Flechtner lists three available graphical front ends for nmap in his Linux Gazette article entitled "Tools of the Trade: nmap"[5]. They are:

NmapFE - NmapFE, written by Zach Smith, comes included in the nmap-2.53.rpm and uses the GTK interface.
NmapFE can be found at http://codebox.net/nmapfe.html

Kmap - Kmap, written by Ian Zepp, uses the QT/KDE front-end for nmap at can be found at
http://www.edotorg.org/kde/kmap/

KNmap - KNmap, written by Alexandre Sagala, is another KDE front-end for nmap and can be found at
http://pages.infinit.net/rewind/

The front ends are useful for quick and dirty scanning when you can't remember a certain parameter but lack the full flexibility of the command line.

DISSECTION OF PORTSCAN

Lets start out with a simple scan. I initiated the following scan from the command line using the following syntax:

```
nmap -oN nmaplog_sS.txt -sS -v 192.168.1.106
```

This is the basic nmap command followed by a series of parameters.
The *–oN* parameter tells nmap to log its output to a file that I specify, in this case, *nmaplog_sS.txt*
The *–sS* parameter tells nmap to use the TCP SYN scan. This is the "half open" scan that is used when the attacker needs to be more stealthy than a full connect scan would allow.
The *–v* parameter tells nmap to be verbose in its output.
The IP address at the end of the command string is the address of the targeted system.

This nmap run produced the following logfile.

```
# nmap (V. 2.54BETA30) scan initiated Wed Oct 17 21:56:32 2001 as: nmap
-oN nmaplog_sS.txt -sS -v 192.168.1.106
Interesting ports on  (192.168.1.106):
(The 1542 ports scanned but not shown below are in state: closed)
Port       State       Service
25/tcp     open        smtp
111/tcp    open        sunrpc
113/tcp    open        auth
515/tcp    open        printer
587/tcp    open        submission
1024/tcp   open        kdm
6000/tcp   open        X11


# Nmap run completed at Wed Oct 17 21:57:53 2001 -- 1 IP address (1
host up) scanned in 81 seconds
```

This scan generates a fair amount of network activity. I didn't specify a range of ports to scan so nmap used its default port setting which is ports between 1 and 1024, and whatever ports are listed in the services file that comes with the nmap distribution. So we are generating over 1000 packets from the nmap scanning system plus the responses from the target.

Lets review how the half-open scan works. A normal TCP connection begins with a three-way handshake between the initiating and receiving systems. The calling system sends a SYN packet, the called system responds with a SYN-ACK, and the calling system answers that with an ACK.

This is how the handshake works if the called system is in fact providing a service on the called port. If not, it sends an RST (reset) instead of the SYN-ACK.

In the half-open scan nmap begins with two PING packets, a traditional ICMP ping and a TCP ACK packet to port 80. The reason for the two pings is to confirm that the target host is alive and reachable. The second packet (to port 80) is sent because many sites will filter ICMP, therefore reducing the effectiveness of the traditional ping. Port 80 is the registered port for web traffic and is usually unfiltered.

This process is illustrated in the packet capture shown below. The attacking system generates a PING in frame 1, followed by an ACK packet sent to port 80. This ACK packet has a randomly chosen source port of 60070.

. Frame	Source Address	Dest. Address	Size	Summary
1	[192.168.1.101]	[192.168.1.106]	60	ICMP: Echo
2	[192.168.1.101]	[192.168.1.106]	60	TCP: D=80 S=60070 ACK=2925659581 WIN=1024
3	[192.168.1.106]	[192.168.1.101]	60	ICMP: Echo reply
4	[192.168.1.106]	[192.168.1.101]	60	TCP: D=60070 S=80 RST WIN=0

The target system replies to the PING in frame 3 and sends the RST from port 80 back to port 60070 in frame 4. Again, some systems will block ICMP PINGs, so we send the TCP ACK packet to be thorough

Once nmap determines the host is up it begins sending SYN packets to the target ports. If a SYN-ACK is received this tells the scanner that the port is open on the target. In this case a RST is sent by the scanner to the target to tear down the connection before it is actually established. If the target port is not listening the target machine responds with an RST.

A sample capture of this portion of the attack is shown below. I selected this group of frames to illustrate the random order of ports scanned, the random selection of the source port, and the behavior of the attacker and targets with open and closed target ports.

Frame	Source Address	Dest. Address	Summary
1	[192.168.1.101]	[192.168.1.106	TCP: D=763 S=60050 SYN SEQ=3879399063 LEN=0 WIN=1024
2	[192.168.1.106]	[192.168.1.101	TCP: D=60050 S=763 RST ACK=3879399064 WIN=0
3	[192.168.1.101]	[192.168.1.106	TCP: D=161 S=60050 SYN SEQ=3879399063 LEN=0 WIN=1024
4	[192.168.1.101]	[192.168.1.106	TCP: D=785 S=60050 SYN SEQ=3879399063 LEN=0 WIN=1024
5	[192.168.1.101]	[192.168.1.106	TCP: D=2067 S=60050 SYN SEQ=3879399063 LEN=0 WIN=1024
6	[192.168.1.101]	[192.168.1.106	TCP: D=448 S=60050 SYN SEQ=3879399063 LEN=0 WIN=1024
7	[192.168.1.106]	[192.168.1.101	TCP: D=60050 S=161 RST ACK=3879399064 WIN=0
8	[192.168.1.106]	[192.168.1.101	TCP: D=60050 S=785 RST ACK=3879399064 WIN=0
9	[192.168.1.106]	[192.168.1.101	TCP: D=60050 S=2067 RST ACK=3879399064 WIN=0
10	[192.168.1.106]	[192.168.1.101	TCP: D=60050 S=448 RST ACK=3879399064 WIN=0
11	[192.168.1.101]	[192.168.1.106	TCP: D=6000 S=60050 SYN SEQ=3879399063 LEN=0 WIN=1024
12	[192.168.1.106]	[192.168.1.101	TCP: D=60050 S=6000 SYN ACK=3879399064 SEQ=1810531748 LEN=0 WIN=32696
13	[192.168.1.101]	[192.168.1.106	TCP: D=6000 S=60050 RST WIN=0

Nmap scans the target port list (either provided in the command line or the nmap default – ports 1 – 1024 plus whatever ports are listed in the nmap distributions services file) using a randomly chosen source port. It scans the target ports in random order.

In frame 1 above nmap sends the SYN to port 763. In frame 2 the target responds with an RST because the port is not listening. Frames 3, 4, 5, and 6 send SYN packets to ports 161, 785, 2067, and 448 respectively. They are answered with RSTs in frames 7, 8, 9, and 10 indicating that those ports are also not active.

In frame 11 nmap sends a SYN packet to port 6000, which is active on the target system. The target responds with a SYN-ACK in frame 12. Nmap immediately responds with an RST ending the connection. We have deduced the port is listening and do not need the full connection that would have been established had the scanning system responded with the expected ACK. The reason, again, that we do not respond with the ACK is that the half-open scan may not be logged by the target while the full connection probably would be.

OUTPUT OF FIREWALL LOG

I ran a similar attack against a host being protected by the free version of ZoneAlarm personal firewall. The specific nmap syntax I used was

nmap –v –sS –P0 –oN attackzonealarm.txt 192.168.1.108

The specific meanings of the parameters are –v for verbose output, -sS for a SYN scan, -P0 telling nmap not to ping the host first, -oN attackzonealarm.txt to log the output to the file attackzonealarm.txt, and finally the IP address to scan. The –P0 tells nmap to scan the host even though it does not receive a ping response from the host to confirm that it is up.

This command syntax produces the output file:

```
# nmap (V. 2.54BETA30) scan initiated Thu Oct 25 00:17:13 2001 as: nmap
-v -sS -P0 -oN attackzonealarm.txt 192.168.1.108
All 1549 scanned ports on  (192.168.1.108) are: filtered

# Nmap run completed at Thu Oct 25 00:45:07 2001 -- 1 IP address (1
host up) scanned in 1674 seconds
```

The following is the first few lines of the Zonealrm logfile generated during this scan:

```
ZoneAlarm Logging Client v2.6.231
Windows 98-4.10.1998- -SP
type,date,time,source,destination,transport
FWIN,2001/10/24,19:46:34 -4:00 GMT,192.168.1.101:0,192.168.1.108:0,ICMP
(type:8/subtype:0)
FWOUT,2001/10/24,19:48:42 -4:00 GMT,192.168.1.108:1027,192.168.1.102:139,TCP
(flags:S)
FWOUT,2001/10/24,19:50:20 -4:00 GMT,192.168.1.108:1028,192.168.1.102:139,TCP
(flags:S)
FWOUT,2001/10/24,19:51:58 -4:00 GMT,192.168.1.108:1029,192.168.1.102:139,TCP
(flags:S)
FWIN,2001/10/24,19:52:03 -4:00 GMT,192.168.1.101:52736,192.168.1.108:266,TCP
(flags:S)
FWIN,2001/10/24,19:52:03 -4:00 GMT,192.168.1.101:52736,192.168.1.108:1456,TCP
(flags:S)
FWIN,2001/10/24,19:52:03 -4:00 GMT,192.168.1.101:52736,192.168.1.108:1498,TCP
(flags:S)
FWIN,2001/10/24,19:52:03 -4:00 GMT,192.168.1.101:52736,192.168.1.108:971,TCP
(flags:S)
FWIN,2001/10/24,19:52:03 -4:00 GMT,192.168.1.101:52736,192.168.1.108:140,TCP
(flags:S)
FWIN,2001/10/24,19:52:03 -4:00 GMT,192.168.1.101:52736,192.168.1.108:876,TCP
(flags:S)
FWIN,2001/10/24,19:52:03 -4:00 GMT,192.168.1.101:52736,192.168.1.108:364,TCP
(flags:S)
FWIN,2001/10/24,19:52:03 -4:00 GMT,192.168.1.101:52736,192.168.1.108:86,TCP
(flags:S)
FWIN,2001/10/24,19:52:03 -4:00 GMT,192.168.1.101:52736,192.168.1.108:619,TCP
(flags:S)
```

```
FWIN,2001/10/24,19:52:03 -4:00 GMT,192.168.1.101:52736,192.168.1.108:424,TCP
(flags:S)
```

Looking at the ZoneAlarm output we can see the random order in which the ports are probed, as mentioned earlier. We can also see that in this instance ZoneAlarm prevented the scan from providing any useful information to the attacker.

This information is not intended to make any qualitative assessment of ZoneAlarm or its ability to protect against any form of attack or information gathering. I merely desire to illustrate what the log shows during a scan.

For information purposes I would like to take a look at a packet capture of an nmap scan when the target host is screened.

I ran:

nmap -v -sS -P0 -oN nmapscreen.txt -p 1-5 -r 192.168.1.108

I introduced a couple of changes into the parameters used here to help illustrate the behavior of nmap. I limited the number of ports scanned to 1 through 5 to keep us from being overwhelmed by output and also had the ports scanned sequentially so that it would be easier to follow along in the capture.

So the options translate to this, -v for verbose, -sS for the SYN scan, -P0 to scan even if there Is no ping response, -oN nmapscreen.txt to save the output of nmap to a text file, -p 1-5 to limit the ports scanned to 1 through 5, -r to not randomize the order the ports are scanned in, and finally the target IP address.

Below is an excerpt from the packet capture of the scan initiated by this command.

```
Frame Source Address    Dest. Address    Summary
   1  [192.168.1.101]  [192.168.1.108  TCP: D=1 S=1078 SYN SEQ=1678322423 LEN=0 WIN=32120
   2  [192.168.1.101]  [192.168.1.108]  TCP: D=2 S=1079 SYN SEQ=1686488603 LEN=0 WIN=32120
   3  [192.168.1.101]  [192.168.1.108]  TCP: D=3 S=1080 SYN SEQ=1690785611 LEN=0 WIN=32120
   4  [192.168.1.101]  [192.168.1.108]  TCP: D=4 S=1081 SYN SEQ=1692222671 LEN=0 WIN=32120
   5  [192.168.1.101]  [192.168.1.108]  TCP: D=5 S=1082 SYN SEQ=1682102237 LEN=0 WIN=32120
   6  [192.168.1.101]  [192.168.1.108]  TCP: D=1 S=1078 SYN (Retransmission of Frame 1) SEQ=1678322423 LEN=0
WIN=32120
   7  [192.168.1.101]  [192.168.1.108]  TCP: D=2 S=1079 SYN (Retransmission of Frame 2) SEQ=1686488603 LEN=0
WIN=32120
   8  [192.168.1.101]  [192.168.1.108]  TCP: D=3 S=1080 SYN (Retransmission of Frame 3) SEQ=1690785611 LEN=0
WIN=32120
   9  [192.168.1.101]  [192.168.1.108]  TCP: D=4 S=1081 SYN (Retransmission of Frame 4) SEQ=1692222671 LEN=0
WIN=32120
  10  [192.168.1.101]  [192.168.1.108]  TCP: D=5 S=1082 SYN (Retransmission of Frame 5) SEQ=1682102237 LEN=0
WIN=32120
  11  [192.168.1.101]  [192.168.1.108]  TCP: D=1 S=1083 SYN SEQ=1691935724 LEN=0 WIN=32120
  12  [192.168.1.101]  [192.168.1.108]  TCP: D=2 S=1084 SYN SEQ=1689017580 LEN=0 WIN=32120
  13  [192.168.1.101]  [192.168.1.108]  TCP: D=3 S=1085 SYN SEQ=1685468911 LEN=0 WIN=32120
  14  [192.168.1.101]  [192.168.1.108]  TCP: D=4 S=1086 SYN SEQ=1695204490 LEN=0 WIN=32120
  15  [192.168.1.101]  [192.168.1.108]  TCP: D=5 S=1087 SYN SEQ=1684340413 LEN=0 WIN=32120
  16  [192.168.1.101]  [192.168.1.108]  TCP: D=1 S=1078 SYN (Retransmission of Frame 1) SEQ=1678322423 LEN=0
WIN=32120
  17  [192.168.1.101]  [192.168.1.108]  TCP: D=2 S=1079 SYN (Retransmission of Frame 2) SEQ=1686488603 LEN=0
WIN=32120
  18  [192.168.1.101]  [192.168.1.108]  TCP: D=3 S=1080 SYN (Retransmission of Frame 3) SEQ=1690785611 LEN=0
WIN=32120
```

19 [192.168.1.101] [192.168.1.108] TCP: D=4 S=1081 SYN (Retransmission of Frame 4) SEQ=1692222671 LEN=0 WIN=32120
20 [192.168.1.101] [192.168.1.108] TCP: D=5 S=1082 SYN (Retransmission of Frame 5) SEQ=1682102237 LEN=0 WIN=32120
21 [192.168.1.101] [192.168.1.108] TCP: D=1 S=1083 SYN (Retransmission of Frame 11) SEQ=1691935724 LEN=0 WIN=32120
22 [192.168.1.101] [192.168.1.108] TCP: D=2 S=1084 SYN (Retransmission of Frame 12) SEQ=1689017580 LEN=0 WIN=32120
23 [192.168.1.101] [192.168.1.108] TCP: D=3 S=1085 SYN (Retransmission of Frame 13) SEQ=1685468911 LEN=0 WIN=32120
24 [192.168.1.101] [192.168.1.108] TCP: D=4 S=1086 SYN (Retransmission of Frame 14) SEQ=1695204490 LEN=0 WIN=32120
25 [192.168.1.101] [192.168.1.108] TCP: D=5 S=1087 SYN (Retransmission of Frame 15) SEQ=1684340413 LEN=0 WIN=32120
26 [192.168.1.101] [192.168.1.108] TCP: D=1 S=1088 SYN SEQ=1697522397 LEN=0 WIN=32120
27 [192.168.1.101] [192.168.1.108] TCP: D=2 S=1089 SYN SEQ=1699420134 LEN=0 WIN=32120
28 [192.168.1.101] [192.168.1.108] TCP: D=3 S=1090 SYN SEQ=1693908666 LEN=0 WIN=32120
29 [192.168.1.101] [192.168.1.108] TCP: D=4 S=1091 SYN SEQ=1692174637 LEN=0 WIN=32120
30 [192.168.1.101] [192.168.1.108] TCP: D=5 S=1092 SYN SEQ=1692065415 LEN=0 WIN=32120
31 [192.168.1.101] [192.168.1.108] TCP: D=1 S=1083 SYN (Retransmission of Frame 11) SEQ=1691935724 LEN=0 WIN=32120
32 [192.168.1.101] [192.168.1.108] TCP: D=2 S=1084 SYN (Retransmission of Frame 12) SEQ=1689017580 LEN=0 WIN=32120
33 [192.168.1.101] [192.168.1.108] TCP: D=3 S=1085 SYN (Retransmission of Frame 13) SEQ=1685468911 LEN=0 WIN=32120
34 [192.168.1.101] [192.168.1.108] TCP: D=4 S=1086 SYN (Retransmission of Frame 14) SEQ=1695204490 LEN=0 WIN=32120
35 [192.168.1.101] [192.168.1.108] TCP: D=5 S=1087 SYN (Retransmission of Frame 15) SEQ=1684340413 LEN=0 WIN=32120
36 [192.168.1.101] [192.168.1.108] TCP: D=1 S=1088 SYN (Retransmission of Frame 26) SEQ=1697522397 LEN=0 WIN=32120
37 [192.168.1.101] [192.168.1.108] TCP: D=2 S=1089 SYN (Retransmission of Frame 27) SEQ=1699420134 LEN=0 WIN=32120
38 [192.168.1.101] [192.168.1.108] TCP: D=3 S=1090 SYN (Retransmission of Frame 28) SEQ=1693908666 LEN=0 WIN=32120
39 [192.168.1.101] [192.168.1.108] TCP: D=4 S=1091 SYN (Retransmission of Frame 29) SEQ=1692174637 LEN=0 WIN=32120
40 [192.168.1.101] [192.168.1.108] TCP: D=5 S=1092 SYN (Retransmission of Frame 30) SEQ=1692065415 LEN=0 WIN=32120
41 [192.168.1.101] [192.168.1.108] TCP: D=5 S=1093 SYN SEQ=1707283723 LEN=0 WIN=32120
42 [192.168.1.101] [192.168.1.108] TCP: D=4 S=1094 SYN SEQ=1708428928 LEN=0 WIN=32120
43 [192.168.1.101] [192.168.1.108] TCP: D=3 S=1095 SYN SEQ=1706906126 LEN=0 WIN=32120
44 [192.168.1.101] [192.168.1.108] TCP: D=2 S=1096 SYN SEQ=1710815586 LEN=0 WIN=32120
45 [192.168.1.101] [192.168.1.108] TCP: D=1 S=1097 SYN SEQ=1709275972 LEN=0 WIN=32120
46 [192.168.1.101] [192.168.1.108] TCP: D=5 S=1093 SYN (Retransmission of Frame 41) SEQ=1707283723 LEN=0 WIN=32120
47 [192.168.1.101] [192.168.1.108] TCP: D=4 S=1094 SYN (Retransmission of Frame 42) SEQ=1708428928 LEN=0 WIN=32120
48 [192.168.1.101] [192.168.1.108] TCP: D=3 S=1095 SYN (Retransmission of Frame 43) SEQ=1706906126 LEN=0 WIN=32120
49 [192.168.1.101] [192.168.1.108] TCP: D=2 S=1096 SYN (Retransmission of Frame 44) SEQ=1710815586 LEN=0 WIN=32120
50 [192.168.1.101] [192.168.1.108] TCP: D=1 S=1097 SYN (Retransmission of Frame 45) SEQ=1709275972 LEN=0 WIN=32120

As you can see in the first five packets nmap chooses a starting source port at random and sends a packet to the first destination port. It increments the source port by one and sends to the second target port, and so on.

Remember this target host is screened. It will not respond to the SYN packet. nmap does not know whether the host is failing to respond because it is screened or because the network is unreliable. So in packets 6 through 10 it retransmits the first 5 packets.

In packet 11 it increments the source port used in packet 1 and sends to the first target port all over again. Increments the source port for packet 2 and sends to the second target port, increments the source port of packet 3 and sends to the third target port, etc.

Based on my observations it appears that nmap will retransmit a packet twice (for a total of three transmissions – the original plus two retransmissions). It will increment the source port and try again with a fresh packet after the first retran. The fresh packet will follow the same retransmission and fresh packet incrementation as the original.

According to Fyodor, nmap keeps statistics to gauge the reliability of the network. It will tailor its behavior in relation to retransmissions accordingly.

DECOY

Next I would like to look at the mechanics of an attack using multiple decoys. For the sake of continuity I will use the same basic SYN scan that we have used previously, but with a minor modification or two.

nmap -v -sS -P0 –p 1–25 -oN decoyattackza.txt –r –D192.168.1.100,192.168.1.106, ME 192.168.1.108

This command executes nmap with –v for verbose, -sS for the SYN scan, -p 1-140 to limit the scanned ports to 1-25 (to limit the length of time required to run the scan and the relative size of the output files), -oNdecoyattackza.txt to name the output file, -r to not randomize the target port order, -D to tell nmap to use the list of decoys (192.168.1.100 and 106) and finally the IP address of the victim.

Frame	Destination	Source	Protocol	Summary
1	[192.168.1.108]	[192.168.1.100]	TCP	D=1 S=57592 SYN SEQ=3343476181 LEN=0 WIN=4096
2	[192.168.1.108]	[192.168.1.106]	TCP	D=1 S=57592 SYN (Retransmission of Frame 1) SEQ=3343476181 LEN=0 WIN=4096
3	[192.168.1.108]	[192.168.1.101]	TCP	D=1 S=57592 SYN (Retransmission of Frame 1) SEQ=3343476181 LEN=0 WIN=4096
4	[192.168.1.108]	[192.168.1.100]	TCP	D=2 S=57592 SYN SEQ=3343476181 LEN=0 WIN=4096
5	[192.168.1.108]	[192.168.1.106]	TCP	D=2 S=57592 SYN (Retransmission of Frame 4) SEQ=3343476181 LEN=0 WIN=4096
6	192.168.1.108]	[192.168.1.101]	TCP	D=2 S=57592 SYN (Retransmission of Frame 4) SEQ=3343476181 LEN=0 WIN=4096
7	[192.168.1.108]	[192.168.1.100]	TCP	D=3 S=57592 SYN SEQ=3343476181 LEN=0 WIN=4096
8	[192.168.1.108]	[192.168.1.106]	TCP	D=3 S=57592 SYN (Retransmission of Frame 7) SEQ=3343476181 LEN=0 WIN=4096
9	[192.168.1.108]	[192.168.1.101]	TCP	D=3 S=57592 SYN (Retransmission of Frame 7) SEQ=3343476181 LEN=0 WIN=4096
10	[192.168.1.108]	[192.168.1.100]	TCP	D=4 S=57592 SYN SEQ=3343476181 LEN=0 WIN=4096
11	[192.168.1.108]	[192.168.1.106]	TCP	D=4 S=57592 SYN (Retransmission of Frame 10) SEQ=3343476181 LEN=0 WIN=4096
12	[192.168.1.100]	[192.168.1.108]	TCP	D=57592 S=3 RST ACK=3343476182 WIN=0
13	[192.168.1.108]	[192.168.1.101]	TCP	D=4 S=57592 SYN (Retransmission of Frame 10) SEQ=3343476181 LEN=0 WIN=4096
14	[192.168.1.108]	[192.168.1.100]	TCP	D=5 S=57592 SYN SEQ=3343476181 LEN=0 WIN=4096
15	[192.168.1.108]	[192.168.1.106]	TCP	D=5 S=57592 SYN (Retransmission of Frame 14) SEQ=3343476181 LEN=0 WIN=4096
16	[192.168.1.101]	[192.168.1.108]	TCP	D=57592 S=3 RST ACK=3343476182 WIN=0

```
 17 [192.168.1.100]     [192.168.1.108]  TCP   D=57592 S=4 RST ACK=3343476182 WIN=0
 18 [192.168.1.108]     [192.168.1.101]  TCP   D=5 S=57592 SYN (Retransmission of Frame 14) SEQ=3343476181 LEN=0
WIN=4096
 19 [192.168.1.106]     [192.168.1.108]  TCP   D=57592 S=4 RST ACK=3343476182 WIN=0
 20 [192.168.1.108]     [192.168.1.100]  TCP   D=6 S=57592 SYN SEQ=3343476181 LEN=0 WIN=4096
 21[192.168.1.101]     [192.168.1.108]   TCP   D=57592 S=4 RST ACK=3343476182 WIN=0
 22 [192.168.1.100]     [192.168.1.108]  TCP   D=57592 S=5 RST ACK=3343476182 WIN=0
 23 [192.168.1.106]     [192.168.1.108]  TCP   D=57592 S=5 RST ACK=3343476182 WIN=0
 24[192.168.1.108]     [192.168.1.106]   TCP   D=6 S=57592 SYN (Retransmission of Frame 20) SEQ=3343476181 LEN=0
WIN=4096
 25[192.168.1.108]     [192.168.1.101]   TCP   D=6 S=57592 SYN (Retransmission of Frame 20) SEQ=3343476181 LEN=0
WIN=4096
 26 [192.168.1.101]     [192.168.1.108]  TCP   D=57592 S=5 RST ACK=3343476182 WIN=0
 27 [192.168.1.100]     [192.168.1.108]  TCP   D=57592 S=6 RST ACK=3343476182 WIN=0
 28 [192.168.1.106]     [192.168.1.108]  TCP   D=57592 S=6 RST ACK=3343476182 WIN=0
 29 [192.168.1.108]     [192.168.1.100]  TCP   D=7 S=57592 SYN SEQ=3343476181 LEN=0 WIN=4096
 30  [192.168.1.108]     [192.168.1.106]  TCP   D=7 S=57592 SYN (Retransmission of Frame 29) SEQ=3343476181 LEN=0
WIN=4096
 31  [192.168.1.101]     [192.168.1.108]  TCP   D=57592 S=6 RST ACK=3343476182 WIN=0
 32  [192.168.1.108]     [192.168.1.101]  TCP   D=7 S=57592 SYN (Retransmission of Frame 29) SEQ=3343476181 LEN=0
WIN=4096
```

The output is divided into several columns. From left to right they are packet number, destination IP address, source IP address, protocol type, and summary. The summary field begins with the source and destination port numbers followed by the type of TCP packet it is (SYN, ACK or RST).

This scan behaves exactly like the earlier SYN scans we have seen. The scanning station sends a SYN to the target port. If open the target port responds with a SYN/ACK. If closed the target port responds with an RST. If the scanner receives a SYN/ACK it immediately tears down the connection by sending an RST of its own back to the target port.

The key difference with this type of attack is the obfuscation of the source of the attack by the generation of the additional scans using the decoy IP addresses.

As you can see the scan begins with target port 1. We placed ourselves 3rd in the decoy list (see the ME in the list of decoys following the –D in the command line). Three packets are generated with port 1 as the target port. One from source IP 192.168.1.100, one from 192.168.1.106 and one from 192.168.1.101 which is the actual scanning device. Please understand that all three packets were in fact generated by the same device. We are simply manipulating the source IP address field to reflect those hosts listed in our decoy list.

This can be illustrated by looking at a detailed packet analysis of the first two packets. Please notice that frame 1 and frame 2 are identical except for the source IP address field and the DLC source field. The scanner can manipulate the IP address but its physical point of origin will always be the DLC source (or MAC address) of its own network interface card. If you look at the DLC headers for each frame and find the line for the DLC source you will see that the address is Sun A28383. This shows that even though the apparent source IP address is different in each frame they, in fact, originated on the same device.

-------------------- Frame 1 --------------------

DLC: ----- DLC Header -----

```
     DLC:
     DLC:  Frame 1 arrived at  13:17:20.7107; frame size is 60 (003C hex) bytes.
     DLC:  Destination = Station 0050DAB1BAB8
     DLC:  Source      = Station Sun   A28383
     DLC:  Ethertype   = 0800 (IP)
     DLC:
IP: ----- IP Header -----
     IP:
     IP: Version = 4, header length = 20 bytes
     IP: Type of service = 00
     IP:       000. ....   = routine
     IP:       ...0 .... = normal delay
     IP:       .... 0... = normal throughput
     IP:       .... .0.. = normal reliability
     IP:       .... ..0. = ECT bit - transport protocol will ignore the CE bit
     IP:       .... ...0 = CE bit - no congestion
     IP: Total length    = 40 bytes
     IP: Identification  = 21421
     IP: Flags           = 0X
     IP:       .0.. .... = may fragment
     IP:       ..0. .... = last fragment
     IP: Fragment offset = 0 bytes
     IP: Time to live    = 43 seconds/hops
     IP: Protocol        = 6 (TCP)
     IP: Header checksum = B802 (correct)
     IP: Source address      = [192.168.1.100]
     IP: Destination address = [192.168.1.108]
     IP: No options
     IP:
TCP: ----- TCP header -----
     TCP:
     TCP: Source port            = 57592
     TCP: Destination port       = 1 (TCPmux)
     TCP: Initial sequence number = 3343476181
     TCP: Next expected Seq number= 3343476182
     TCP: Data offset            = 20 bytes
     TCP: Flags                  = 02
     TCP:              ..0. .... = (No urgent pointer)
     TCP:              ...0 .... = (No acknowledgment)
     TCP:              .... 0... = (No push)
     TCP:              .... .0.. = (No reset)
     TCP:              .... ..1. = SYN
     TCP:              .... ...0 = (No FIN)
     TCP: Window                 = 4096
     TCP: Checksum               = 0DA9 (correct)
     TCP: No TCP options
     TCP:

- - - - - - - - - - - - - - - - - - - - Frame 2 - - - - - - - - - - - - - - - - - - - -
DLC:  ----- DLC Header -----
     DLC:
     DLC:  Frame 2 arrived at  13:17:20.7108; frame size is 60 (003C hex) bytes.
     DLC:  Destination = Station 0050DAB1BAB8
     DLC:  Source      = Station Sun   A28383
     DLC:  Ethertype   = 0800 (IP)
     DLC:
IP: ----- IP Header -----
     IP:
     IP: Version = 4, header length = 20 bytes
     IP: Type of service = 00
     IP:       000. ....   = routine
     IP:       ...0 .... = normal delay
     IP:       .... 0... = normal throughput
     IP:       .... .0.. = normal reliability
     IP:       .... ..0. = ECT bit - transport protocol will ignore the CE bit
     IP:       .... ...0 = CE bit - no congestion
     IP: Total length    = 40 bytes
     IP: Identification  = 41747
     IP: Flags           = 0X
     IP:       .0.. .... = may fragment
```

```
    IP:       ..0. .... = last fragment
    IP: Fragment offset = 0 bytes
    IP: Time to live    = 43 seconds/hops
    IP: Protocol        = 6 (TCP)
    IP: Header checksum = 6896 (correct)
    IP: Source address      = [192.168.1.106]
    IP: Destination address = [192.168.1.108]
    IP: No options
    IP:
TCP: ----- TCP header -----
    TCP:
    TCP: Source port           = 57592
    TCP: Destination port       = 1 (TCPmux)
    TCP: Initial sequence number = 334347618 1
    TCP: Next expected Seq number= 334347618 2
    TCP: Data offset           = 20 bytes
    TCP: Flags                 = 02
    TCP:             ..0. .... = (No urgent pointer)
    TCP:             ...0 .... = (No acknowledgment)
    TCP:             .... 0... = (No push)
    TCP:             .... .0.. = (No reset)
    TCP:             .... ..1. = SYN
    TCP:             .... ...0 = (No FIN)
    TCP: Window                = 4096
    TCP: Checksum              = 0DA3 (correct)
    TCP: No TCP options
    TCP:
```

The packet decoder itself is not fooled by the manipulated IP addresses. It calls packets 2 and 3 retransmissions of packet 1. It sees all 3 packets originating from the same physical device with identical target addresses and ports, source ports, sequence numbers, etc.

This information cannot be used to distinguish the true source of a decoy attack on a network being attacked across the internet because the DLC or MAC address of the packet reaching the target system will be that of the last router interface it passed through on its trip from IP source to IP destination address.

PORT SCANNING AS A DEFENSIVE TOOL

Port scanning has a definite place in the arsenal of the corporate security officer or concerned system administrator. Periodic scans of your own network and hosts will keep you up to date on what information may be available to outside observers, information gatherers, and potential attackers. To anticipate their activities you must know what they know. You must be able to determine what information is leaking out so that you can evaluate the potential risks and determine the appropriate corrective actions.

nmap can also help you to identify the services that are active on each of your systems. Armed with this information you can close or disable those services not required for a given system to fulfill its mission. If you are not required to provide a service on a given system then it should be turned off.

THE CURRENT LEGAL RAMIFICATIONS OF PORTSCANNING

There is a great debate raging today about the legality of portscanning and whether or not someone who conducts a portscan against a network they do not own should be prosecuted. Most arguments break down into the "portscanning is the same as rattling the doors and windows of a business or home"analogy. The issue is cloudy. The final answer is not yet in.

However in the current law enforcement climate it is probably best ***not*** to scan any host or network other than your own.

SUMMARY

nmap is an excellent, multi functional utility that should be a part of every system administrator's toolkit. It is well documented, open source, widely available and, best of all, free.

It provides a rich set of command line options allowing it to be tailored to many jobs. It provides an administrator with a window into two of the four nuggets of system security.

The four nuggets are:
1. know thy system
2. principle of least privilege
3. prevention is ideal, detection is a must
4. defense in depth

nmap assists the system administrator with nuggets 1 and 2, know thy system and principle of least privilege. Run nmap against your hosts. See what information can be gleaned. Make sure that you know what someone else can learn about your network and devices. Determine what services are being offered on each host and turn off everything not explicitly needed. This is the principle of least privilege. If you don't need it turn it off.

CITATIONS

1. Fyodor, "Remote OS detection via TCP/IP Stack FingerPrinting" April 10, 1999 http://www.insecure.org/nmap/nmap-fingerprinting-article.html (Sept 18th 2000)

2. jvfields@camden411.com, http://www.camden411.com/tcpipfaq/ports.html

3. Rik Farrow, "System Fingerprinting With Nmap" Network Magazine Nov 6, 2000

4. Stuart McClure, Joel Scambray, George Kurtz, HACKING EXPOSED Osborne, McGraw-Hill, 1999

5. Josh Flechtner "Tools of the Trade: nmap" August 2000, Issue 56, Linux Gazette, Published by Linux Journal

RESOURCES

Cole, Eric. HACKERS BEWARE New Riders Publishing, August 2001

lamontg@raven.genome.washington.edu Nmap Guide Sun Apr 11 03:03:15 1999
http://www.insecure.org/nmap/lamont-nmap-guide.txt

Fyodor "The Art of Port Scanning" Phrack Magazine Volume 7, Issue 51 September 01, 1997, article 11 of 17
http://www.insecure.org/nmap/p51-11.txt

Last Updated: December 7th, 2019

Upcoming SANS Training

Click here to view a list of all SANS Courses

SANS Cyber Defense Initiative 2019	**Washington, DCUS**	**Dec 10, 2019 - Dec 17, 2019**	**Live Event**
SANS Austin Winter 2020	**Austin, TXUS**	**Jan 06, 2020 - Jan 11, 2020**	**Live Event**
SANS Miami 2020	**Miami, FLUS**	**Jan 13, 2020 - Jan 18, 2020**	**Live Event**
SANS Threat Hunting & IR Europe Summit & Training 2020	**London, GB**	**Jan 13, 2020 - Jan 19, 2020**	**Live Event**
Cyber Threat Intelligence Summit & Training 2020	**Arlington, VAUS**	**Jan 20, 2020 - Jan 27, 2020**	**Live Event**
SANS Amsterdam January 2020	**Amsterdam, NL**	**Jan 20, 2020 - Jan 25, 2020**	**Live Event**
SANS Tokyo January 2020	**Tokyo, JP**	**Jan 20, 2020 - Jan 25, 2020**	**Live Event**
SANS Anaheim 2020	**Anaheim, CAUS**	**Jan 20, 2020 - Jan 25, 2020**	**Live Event**
MGT521 Beta Two 2020	**San Diego, CAUS**	**Jan 22, 2020 - Jan 23, 2020**	**Live Event**
SANS Vienna January 2020	**Vienna, AT**	**Jan 27, 2020 - Feb 01, 2020**	**Live Event**
SANS Las Vegas 2020	**Las Vegas, NVUS**	**Jan 27, 2020 - Feb 01, 2020**	**Live Event**
SANS San Francisco East Bay 2020	**Emeryville, CAUS**	**Jan 27, 2020 - Feb 01, 2020**	**Live Event**
SANS Security East 2020	**New Orleans, LAUS**	**Feb 01, 2020 - Feb 08, 2020**	**Live Event**
SANS London February 2020	**London, GB**	**Feb 10, 2020 - Feb 15, 2020**	**Live Event**
SANS Northern VA - Fairfax 2020	**Fairfax, VAUS**	**Feb 10, 2020 - Feb 15, 2020**	**Live Event**
SANS New York City Winter 2020	**New York City, NYUS**	**Feb 10, 2020 - Feb 15, 2020**	**Live Event**
SANS Dubai February 2020	**Dubai, AE**	**Feb 15, 2020 - Feb 20, 2020**	**Live Event**
SANS Brussels February 2020	**Brussels, BE**	**Feb 17, 2020 - Feb 22, 2020**	**Live Event**
SANS San Diego 2020	**San Diego, CAUS**	**Feb 17, 2020 - Feb 22, 2020**	**Live Event**
SANS Scottsdale 2020	**Scottsdale, AZUS**	**Feb 17, 2020 - Feb 22, 2020**	**Live Event**
Open-Source Intelligence Summit & Training 2020	**Alexandria, VAUS**	**Feb 18, 2020 - Feb 24, 2020**	**Live Event**
SANS Training at RSA Conference 2020	**San Francisco, CAUS**	**Feb 23, 2020 - Feb 24, 2020**	**Live Event**
SANS Secure India 2020	**Bangalore, IN**	**Feb 24, 2020 - Feb 29, 2020**	**Live Event**
SANS Jacksonville 2020	**Jacksonville, FLUS**	**Feb 24, 2020 - Feb 29, 2020**	**Live Event**
SANS Zurich February 2020	**Zurich, CH**	**Feb 24, 2020 - Feb 29, 2020**	**Live Event**
SANS Manchester February 2020	**Manchester, GB**	**Feb 24, 2020 - Feb 29, 2020**	**Live Event**
ICS Security Summit & Training 2020	**Orlando, FLUS**	**Mar 02, 2020 - Mar 09, 2020**	**Live Event**
Blue Team Summit & Training 2020	**Louisville, KYUS**	**Mar 02, 2020 - Mar 09, 2020**	**Live Event**
SANS Northern VA - Reston Spring 2020	**Reston, VAUS**	**Mar 02, 2020 - Mar 07, 2020**	**Live Event**
SANS Secure Japan 2020	**Tokyo, JP**	**Mar 02, 2020 - Mar 14, 2020**	**Live Event**
SANS Munich March 2020	**Munich, DE**	**Mar 02, 2020 - Mar 07, 2020**	**Live Event**
SANS Frankfurt December 2019	**OnlineDE**	**Dec 09, 2019 - Dec 14, 2019**	**Live Event**
SANS OnDemand	**Books & MP3s OnlyUS**	**Anytime**	**Self Paced**