Interested in learning
more about security?

SANS Institute
InfoSec Reading Room

This paper is from the SANS Institute Reading Room site. Reposting is not permitted without express written permission.

Building an IPv6 Firewall with OpenBSD

IPv6, or IP version 6, has been in development and for several years. It is designed to be the long overdue replacement for IPv4, the current version of IP currently in use across the Internet, and in private networks which are not connected to the Internet. This paper is intended to be a how-to for IPv6 firewalls running on OpenBSD 3.0. It will cover the basics of installing OpenBSD, setting up a tunnel to the 6Bone, and configuring the Packet Filter firewall included with OpenBSD. This paper will not cover IPv6 firew...

Copyright SANS Institute
Author Retains Full Rights

AD
CounterTack
CounterTack Active Forensics
for In-Progress Attacks
GET THE WHITE PAPER NOW >>>

Building an IPv6 Firewall with OpenBSD

by Eric Millican

Introduction

IPv6, or IP version 6, has been in development and for several years. It is designed to be the long overdue replacement for IPv4, the current version of IP currently in use across the Internet, and in private networks which are not connected to the Internet. It touts several features that will make it the protocol of choice over IPv4. The most notable of these features is the use of 128-bit addressing as opposed to the current 32-bit address style in use by IPv4. Other features include better support for mobile users and IP auto configuration.

This paper is intended to be a how-to for IPv6 firewalls running on OpenBSD 3.0. It will cover the basics of installing OpenBSD, setting up a tunnel to the 6Bone, and configuring the Packet Filter firewall included with OpenBSD. This paper will not cover IPv6 firewalls as they apply to mobile IP, but only to hard-wired LANs. The OpenBSD installation will be performed via FTP. It is presumed that the user will have at least some familiarity with IPv4. Familiarity with IPv4 firewalls will also be helpful. I have decided upon OpenBSD 3.0 for two reasons. First is its security track record. Second is the new Packet Filter firewall included with 3.0. Out of all the open source firewalls I have used, it is my opinion that Packet Filter has the best support for IPv6.

What you will need

In order to build your firewall, you will need several things. Obviously, the first requirement is a spare computer that meets the requirements for installing OpenBSD. The computer I will be using for this example has a 75MHz Intel Pentium Processor, 96MB of RAM, a 1 GB hard disk, a 300MB disk, onboard video, and two Linksys ethernet cards (using the "dc" driver). Only one hard disk is required.

Some type of IPv4 Internet connection with a globally routable IP will be required. Fast connectivity is not required. A 56k connection will work, although you may want to consider installing OpenBSD via CD. A static IP is strongly recommended. There are a few IPv6 tunnel brokers that will support a dynamic IP, however, this document will not discuss connectivity to those brokers.

Installing OpenBSD

First you will need to obtain a boot floppy from ftp.openbsd.org. You will find the boot floppies in /pub/OpenBSD/3.0/i386/. There are three different floppies available—floppy30.fs, floppyB30.fs, and floppyC30.fs. Any one of these is capable of installing OpenBSD. Which one you choose depends on your hardware. Each of these floppy images contains some drivers that the other two don't. floppy30.fs is designed to support most desktop systems. It contains support for a wide variety of ethernet devices as well as support for IDE disks and CD-ROMS and support for common SCSI controllers. floppyB30.fs is designed for server systems. This floppy contains drivers for all supported SCSI and RAID controllers as well as gigabit ethernet devices. floppyC30.fs is designed for installation on a laptop. It includes support for PC card and cardbus devices. For more information on OpenBSD's hardware requirements and complete lists of drivers supported by each disk, see http://www.openbsd.org/i386.html.

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

Creating the install disk and beginning installation. For this installation, we will be using floppy30.fs. After retrieving this disk image, it will be necessary to create a floppy disk from the image. If you have an existing Unix/Linux system available, you can use the "dd" utility to create the disk:

 # dd if=floppy28.fs of=/dev/fd0 bs=126b

For Windows systems, you will need to obtain rawrite.exe. This utility can be downloaded from the /pub/OpenBSD/3.0/tools directory on the OpenBSD FTP site. Place it in the same directory you placed the downloaded image file, for convenience. To use rawrite, first open up a command prompt window. Change directory to the location of rawrite and type the command "rawrite". Rawrite will then prompt you for the name of the image file and the drive letter of the drive to which you want to write it. When entering the drive letter, do not use ':'.

Next, insert your newly created install disk into the computer that will become your firewall and boot the PC. You will see various boot messages and a spinning cursor indicating the kernel is being loaded from the disk. After the kernel has been loaded, it will begin to search for hardware. You will see the results of the kernel's search. Before beginning, you will want to make sure your kernel has detected vital hardware (use Shift-Page Up to scroll up). In my case, the following devices were vital:

pciide0 at pci0 dev 3 function 0 "CMD Technology PCI0640" rev 0x02: no DMA, channel 0 wired to compatibility, channel 1 wired to compatibility
wd0 at pciide0 channel 0 drive 0: <M1606TA>
wd0: 32-sector PIO, LBA, 1039MB, 2111 cyl, 16 head, 63 sec, 2128536 sectors
wd1 at pciide0 channel 0 drive 1: <Maxtor 7345 AT>
wd1: 32-sector PIO, CHS, 329MB, 790 cyl, 15 head, 57 sec, 675450 sectors
pciide0: channel 1 ignored (disabled)
dc0 at pci0 dev 14 function 0 "ADMtek AN983" rev 0x11: irq 9 address 00:03:6d:00:3c:6c
ukphy0 at dc0 phy 1: Generic IEEE 802.3u media interface
ukphy0: OUI 0x000895, model 0x0001, rev. 0
dc1 at pci0 dev 15 function 0 "ADMtek AN983" rev 0x11: irq 5 address 00:03:6d:00:3c:64
ukphy1 at dc1 phy 1: Generic IEEE 802.3u media interface
ukphy1: OUI 0x000895, model 0x0001, rev. 0

Next, you will be asked what you want to do. You must select from (I)nstall, (U)pgrade, or (S)hell. The shell option is intended for disaster recovery. Upgrade and Shell require that a previously installed version of OpenBSD be available on the system. We will be performing an install on a fresh system, so select that option. The remainder of this section will provide you with a high level overview of the installation process. It is recommended you review http://www.openbsd.org/faq/faq4.html for more detailed instructions if you are not already familiar with OpenBSD.

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

Partitioning your hard disks. After specifying your terminal type, you will be given the option to partition the space on your hard disks. You will be asked which disk you would like to partition. If you only have one disk, the only option will be wd0. My system has two, so I have wd0 and wd1. I will start with wd0 since it will contain my root partition. The installation program will ask if you would like to use all of the available space for OpenBSD. Since we will not be installing any other OS, select yes. You will then be taken into the partition editor. There are five basic commands we will be using—'a' to add a partition, 'd' to delete a partition (in case an error is made adding one), 'p' to display the partition table, 'w' to write it to disk, and 'q' to quit. Upon entering the partition editor and entering the 'p' command, you will notice that there is already a 'c' partition listed. This is by design. The 'c' partition always occupies the entire disk. Just ignore that it's even there.

Below you will see the partition scheme I have chosen, first for wd0:

8 partitions:

#	size	offset	fstype	[fsize	bsize	cpg]	
a:	204800	63	4.2BSD	1024	8192	16	# /
b:	262144	204863	swap				
c:	2124801	63	unused	0	0		
d:	409600	467007	4.2BSD	1024	8192	16	# /var
e:	1248194	876607	4.2BSD	1024	8192	16	# /usr

And now wd1:

8 partitions:

#	size	offset	fstype	[fsize	bsize	cpg]	
a:	413192	57	4.2BSD	1024	8192	16	# /tmp
b:	262144	413249	swap				
c:	675393	57	unused	0	0		

The size listed above is the number of blocks occupied by that partition. Each block is 512 bytes. This gives us 100MB for /, 128MB for swap space, 200MB for /var, and the remaining space, or about 610MB, for /usr on the first disk. The second disk has partitions for /tmp at approximately 200MB, and another 128MB swap partition. This should give us plenty of swap space for the day to day activities of this system. For a larger site, you may require more swap space. This also gives us a reasonable amount of space in /usr to install software we will need later.

Optional components and configuration. At this point, it may be desirable to install some additional system software or do additional configuration. By creating a .forward file, I have instructed my firewall to forward all of root's mail to another mailbox so that I can receive daily reports from automated tasks. Also, I have decided to setup a serial console because after the installation is complete, there will be no keyboard or monitor attached to this system. Finally, I have chosen to install the bash shell. I find it easier to use than the default shell, primarily because of tab completion. For more information on these and other customizations and software packages, see http://www.openbsd.org. Keep in mind, however, that such software

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

packages may add additional security concerns to your firewall that will not be covered here, so use them at your own risk. You might also want to take the time to look over the information regarding vulnerabilities and patches (if any) for OpenBSD 3.0.

Securing logins. We will now add a non-root user. This user will be a normal user of the system with one exception. The user will be added to the wheel group so that he or she may use su. Users can be added using the adduser tool. It is fairly straightforward and interactive. I have chosen to keep home directories in /usr/home. Since the /usr partition will be mounted read-only, you may wish to have a separate partition for home directories if you intend on placing user files there. After adding your user, edit /etc/group with VI or your favorite text editor. The first line should be "wheel:*:0:root". To the end of the line, add a comma followed by the username for the user you just created.

Next we will disallow all root logins. Before continuing, make sure you can login successfully with the user account you just created and that you can successfully su to root. After disabling root logins, if your user account can't login or can't su to root, you'll need to use the install disk to reboot your system and either re-enable root logins or fix the problem with your user account. Open the file /etc/ttys, once again, in VI or your favorite editor. Remove all occurrences of the word "secure". After exiting the editor, issue the command "*kill –HUP 1*". This will cause init to re-read /etc/ttys. Root logins will now be disabled from the console and any serial lines you have enabled.

At this point, root logins are still allowed via ssh. We're going to change that now. The next file we will edit will be /etc/sshd_config. There are several lines of interest:

1. Uncomment "Protocol 2,1". This will cause sshd to prefer version 2 of the ssh protocol.
2. Change "PermitRootLogin yes" to "PermitRootLogin no". This will cause sshd to disallow root logins. As with the console, you will need to login as a normal user and then su to root.
3. Uncomment "UseLogin no". This option has had recent security vulnerabilities. Though they have been patched, I still don't quite trust it.
4. Uncomment "MaxStartups 10:30:60". This option is useful to protect sshd against syn floods. I highly recommend it.
5. There are other option you may want to change pertaining to the types of encryption and authentication methods used. Consult the sshd(8) man page and your organization's security policy.

You will now need to locate the PID for sshd using the "ps" command. Once you have identified the PID, restart sshd with 'kill –HUP <PID>'.

Secure system services. Edit the file /etc/rc.conf.local. A blank file should be created if this file is not already present. This file is the startup configuration file. It defines which services are enabled at boot time. A list of valid options and their defaults can be found in /etc/rc.conf. Add the following lines to /etc/rc.conf.local:

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

sendmail_flags=NO
portmap=NO
inetd=NO
ntpd=NO

Next, you will need to locate the PID for each of these processes and kill it with the "kill" command. You may specify multiple pids on the kill command line.

Getting connected

Obviously, you will need some way of connecting to the 6Bone. Eventually, IPv6 will be the protocol of choice for the Internet. When this day comes, every cable modem, xDSL, dial-up, and every other type of connection will be using IPv6 instead of IPv4. That day is not here yet. Though there are a few organizations that already provide these native IPv6 services, they are primarily aimed at large organizations that can afford the cost of a T-1 or better. For the small business and average home user, IPv6-over-IPv4 tunnels must be used. These tunnels work by creating a virtual interface on the host or router. In the case of OpenBSD, this interface is called the "gif" interface. IPv6 datagrams sent to this interface are encapsulated in IPv4 datagrams and transmitted to a pre-determined host.

Finding a tunnel broker. So, we know how the gif interface works. But where will we send the encapsulated datagrams? We will have to find someone with 6Bone connectivity who is willing to manage the other end of our tunnel. Enter the tunnel broker. There are many tunnel brokers available across most of the world. Some will require that you e-mail them with your request for a tunnel. Others may have a web based form you can fill out to have a tunnel automatically created. Some will provide you with a block of IP addresses. Some will require you to obtain your own block.

You can find a list of tunnel brokers at http://hs247.com/. You must now decide which one is right for you. First, start out by examining their requirements, if any, and what they offer. It is possible to connect only a single machine with these tunnel brokers. But, for the sake of this example, you will need one that is willing to provide you with a /64 block of IPs. Next, you will want to determine how close these tunnel brokers are to you in IPv4 terms. You can do this with the standard ping or traceroute. If you can locate the broker's tunnel endpoint, use that address. Otherwise, you may have to simply use their website and assume the tunnel endpoint is on the same or a nearby LAN. Keep in mind that a broker geographically close to you might not be the best choice. I have chosen to use British Telecom to establish my tunnel, even though there are tunnel brokers here in the US. Because I am on the East Coast, I often have better connectivity to the UK than to the Midwestern US or to the West Cost.

Setting up the tunnel. Once you have made arrangements with a tunnel broker, you will need to configure your firewall for that tunnel. You will need to have the following information available:

- IPv4 address of the remote endpoint
- IPv6 address of the remote endpoint
- Your endpoint's IPv6 address

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

- Your IPv6 address block

British Telecom provided me with the following:

- Remote IPv4 endpoint: 193.113.58.80
- Remote IPv6 address: 2001:618:400::1:b037:4216
- Local IPv6 address: 2001:618:400::1:b037:4217
- IPv6 address block: 2001:618:400:465::/64
- My local IPv4 address: 216.27.161.11

I'll take a moment to give a brief explanation of IPv6 addresses. The addresses are eight blocks of four hex digits separated by colons. A fully expanded address might be something like "2001:0618:0400:0465:0203:6dff:fe00:3c64". Dropping off leading zero's in each of the blocks gives us something like "2001:618:400:465:203:6dff:fe00:3c64". In addition, we can use "::" to mean "fill in the necessary zero's here". "::" can only be used once. So, "2001:618:400:465:0:0:0:0" becomes "2001:618:400:465::".

Using this information, we will now configure the gif0 interface. Edit the file /etc/hostname.gif0. A blank file should be created. Add the following lines (replacing the addresses with those provided by your tunnel broker):

tunnel 216.27.161.11 193.113.58.80
! ifconfig gif0 inet6 2001:618:400::1:B037:4217 2001:618:400::1:B037:4216 prefixlen 127
! route add -inet6 default 2001:618:400::1:B037:4216

The first line in this file is to establish the IPv4 endpoints. This tells gif0 where to send the encapsulated datagrams and on which local address it should listen for incoming encapsulated datagrams. The exclamation point at the beginning of the second line informs the startup script that we want to execute a command. The command follows the exclamation point. This command configures the local and remote IPv6 addresses and specifies the prefix length. The prefix length is similar in function to IPv4's netmask. A prefix length of 127 indicates that 127 out of the 128 possible bits are used for the network address. It identifies this link as a point-to-point link. Finally, the third line adds our default route to the routing table.

Setting up your subnet

What's a firewall without something to protect. In this section, we will setup the subnet that will exist behind the firewall. It is presumed that all of the physical wiring and hubs/switches are already in place. For this example, I will be using the interface dc1 as my internal interface. You will need to replace dc1 with the name of your interface in the examples. You will also need to replace references to the 2001:618:400:465::/64 address block with the one assigned to you.

Configuring the interface. Create a file called /etc/hostname.dc1. It should contain the following lines:

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

inet6 2001:618:400:465:203:6dff:fe00:3c64 64
inet6 alias 2001:618:400:465:: 64 anycast

2001:618:400:465:203:6dff:fe00:3c64 is the address I have chosen for the internal interface of my firewall. You will want to choose a unique one for yours. But how did I come up with this address? Well, it's mostly arbitrary. I could have just as easily used 2001:618:400:465::1. Each interface has a link local address starting with fe80::. I simply took the first four blocks of my network address and the last four blocks of the link local address and combined them.

The first line above configures dc1 with IPv6 address 2001:618:400:465:203:6dff:fe00:3c64 and a prefix length of 64. The second line adds the anycast address 2001:618:400:465::. Anycast addresses are new to IPv6. They allow two or more devices to share an IP. In theory, this would allow us to have multiple routers, each with the anycast address 2001:618:400:465::. Our workstations would then use 2001:618:400:465:: as their default gateway and whichever router responded first is sent the traffic.

Configure the router advertisement daemon and enabling routing. Rtadvd is the router advertisement daemon. Its job is to respond to router solicitations sent by the workstations. The router advertisement daemon tells the workstation the network address, prefix length, and gateway. The router advertisement daemon's configuration is in /etc/rtadvd.conf. Edit that file, now. You will notice it's blank. This file follows the same format as /etc/termcap. For more options, see the rtadvd.conf(5) man page. You will need to enter the following information:

dc1:
:addrs#1:addr="2001:618:400:465::":prefixlen#64:

Finally, we will need to instruct the system to execute rtadvd at system startup. Once again, edit /etc/rc.conf.local and add the following line:

rtadvd_flags=dc1

This line tells rtadvd which interfaces on which to listen for router solicitations. Each interface must be listed in this line (separated by spaces) and must have an entry in rtadvd.conf.

Next we will enable IP forwarding for IPv6. Edit /etc/sysctl.conf. Uncomment the line containing "net.inet6.ip6.forwarding=1". This will enable forwarding of IPv6 datagrams at boot time.

Applying the changes. Now, we will apply the changes to the network. Execute the command "*sh /etc/netstart*" to configure the interfaces. This will configure and bring up both gif0 and dc1. Verify that they are configured correctly with ifconfig –a. You should have something like the following:

dc1: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

media: Ethernet autoselect (100baseTX full-duplex)
status: active
inet6 fe80::203:6dff:fe00:3c64%dc1 prefixlen 64 scopeid 0x2
inet6 2001:618:400:465:203:6dff:fe00:3c64 prefixlen 64
inet6 2001:618:400:465:: prefixlen 64 anycast gif0: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 1280
physical address inet 216.27.161.11 --> 193.113.58.80
inet6 fe80::203:6dff:fe00:3c6c%6 -> :: prefixlen 64 scopeid 0x6
inet6 2001:618:400::1:b037:4217 -> 2001:618:400::1:b037:4216 prefixlen 127

You should also have a default route configured. Use "*netstat -nr -f inet6*" to show your IPv6 routing table. Next, execute "*sysctl –w net.inet6.ip6.forwarding=1*". You should receive some output indicating the value of net.inet6.ip6.forwarding is changed to 1. Finally, execute "*rtadvd dc1*" to start rtadvd.

To configure an OpenBSD workstation (not a router) to send router solicitations and to accept router advertisements, set "rtsold_flags" equal to the list of IPv6 interfaces you wish to be autoconfigured in /etc/rc.conf.local. In addition, set net.inet6.ip6.forwarding to 0 and net.inet6.ip6.accept_rtadv to 1 in /etc/sysctl.conf. Other BSD based operating systems have similar options. For non-BSD operating systems, read the documentation included with the operating system or the IPv6 stack. Useful information about IPv6 enabled operating systems and third party IPv6 stacks can be found at http://hs247.com/.

Introduction to Packet Filter

Packet Filter is the firewall package that is included with OpenBSD 3.0 and above. It was written to take the place of the IPF firewall that was found in previous versions of OpenBSD. Packet Filter supports many of the options supported by IPF. Packet Filter also has several features that IPF doesn't. It has better support for IPv6, including stateful inspection for TCP, UDP, and ICMPv6 packets. It also has a similar command structure. Those already familiar with IPF should have no trouble converting to packet filter.

Throughout the remainder of this paper, we will discuss packet filter as it applies to our IPv6 firewall. We will discuss the operation of packet filter as it relates to IPv4 only to instruct it to allow traffic across the tunnel. Our firewall will permit inbound traffic on ports we specify and allow all outbound traffic. Stateful inspection will be used on all protocols to help guard against TCP session hijacking, among other things.

Rule processing. PF rules are commonly stored in /etc/pf.conf. Consider the following pf.conf:

pass in all
block in all

What do these rules do when a packet arrives? Packet Filter determines what it should do with a packet based on the last rule which matches that packet. In this case, rule #1 tells PF to pass in all traffic. PF remembers that it's going to pass the packet and moves on to the next rule. Rule #2 tells PF to block all packets. Now, PF thinks it's going to block the packet. It moves on

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

to the next rule. There are no more rules, so PF does what it was told last. In this case, it blocks the packet.

What if we want to alter this behavior? We can! By adding the key word "quick", we can instruct PF to stop processing rules when it finds a match. Consider the following:

pass in quick all
block in all

In this example, PF will pass all traffic. This is because we have used the "quick" key word in rule #1. When PF matches the packet against rule #1, it stops processing rules and passes the packet.

Source and destination. A firewall that does not support matching packets based on source and destination addresses and possibly ports is not a very good firewall. PF uses a very logical "from [source] port = [port] to [source] port = [port]" format to specify source and destination. For example, we could say:

pass in quick inet6 proto tcp from any to FEC0::1 port = 80
pass out quick inet6 proto tcp from FEC0::1 port = 80 to any
block in all

The use of "port = [port]" is not required. If omitted, it simply means "match any port". Since not all IP protocols have a concept of ports, we must specify which protocol we want the rule to match before we can specify a port. In this case, we have specified TCP. We have also specified the address family "inet6", or IPv6, here. This tells PF that we want to match IPv6 packets and that it should be expecting IPv6 addresses for the source and destination. The last surprise in this example was the use of the "out" key word in the second rule. This tells PF to match outbound traffic instead of inbound. These three rules will allow incoming traffic to FEC0::1 on port 80 and the corresponding reply. All other packets will be blocked.

Fooling the attacker. There is one problem with blocking all unwanted traffic. There is no reply sent back to the source. An attacker could guess that there is a firewall in place based on the lack of a response to closed ports. We can alter this behavior with the "return-rst", "return-icmp", and "return-icmp6". "return-rst" returns a TCP reset packet. This would make a blocked port appear to be closed. "return-icmp" and "return-icmp6" return ICMPv4 and ICMPv6 port unreachable messages. Consider the following:

pass in quick inet6 proto tcp from any to FEC0::1 port = 80
pass out quick inet6 proto tcp from FEC0::1 port = 80 to any
block in all
block out all
block return-rst in proto tcp all
block return-rst out proto tcp all
block return-icmp in inet proto udp all
block return-icmp out inet proto udp all

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

block return-icmp6 in inet6 proto udp all
block return-icmp6 out inet6 proto udp all

Now, not only are we blocking packets, but we are also returning messages to the sender to fake it into thinking that nothing is listening on any port except TCP/80. Note that we are responding to both UDP over IPv4 and IPv6. That is because our firewall has a dual-stack. That is it's configured for both IPv4 and IPv6. We want to use PF to protect the firewall itself as well as what's behind the firewall.

Verifying the interface. Next, we will modify the rules to match based on interface. This can be accomplished by adding "via [interface]" to our rules. Later, this will be used to provide egress filtering. Consider the following:

pass in quick on gif0 inet6 proto tcp from any to FEC0::1 port = 80
pass out quick on gif0 inet6 proto tcp from FEC0::1 port = 80 to any
pass in quick on dc1 all
pass out quick on dc1 all
block in all
block out all
block return-rst in proto tcp all
block return-rst out proto tcp all
block return-icmp in inet proto udp all
block return-icmp out inet proto udp all
block return-icmp6 in inet6 proto udp all
block return-icmp6 out inet6 proto udp all

Now, we are only allowing access to our web server from connections arriving on gif0. Rules 3 and 4 have been added to allow all traffic in and out on dc1. Let's assume that dc1 is the network containing our web server and other trusted machines. Without these rules, nothing would be able to reach our web server. The packets would be passed in on gif0, but would be denied on dc1 and a TCP reset would be sent.

Keeping state. The rules so far create a fairly tight firewall. However, we are still prone to TCP hijacking. Also we are allowing all traffic destined for port 80, even somewhat malicious packets. These rules would let packets with illegal flag combinations through. We can solve this problem by modifying the ruleset as such:

pass in quick on gif0 inet6 proto tcp from any to FEC0::1 port = 80 flags S keep state
pass in quick on dc1 all
pass out quick on dc1 all
block in all
block out all
block return-rst in proto tcp all
block return-rst out proto tcp all
block return-icmp in inet proto udp all
block return-icmp out inet proto udp all

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

block return-icmp6 in inet6 proto udp all
block return-icmp6 out inet6 proto udp all

There are two new options introduced here. The first is the "flags" option. By specifying "flags S", we are telling PF that we only want to match TCP packets that have *only* their SYN flag set. The format of this option is "flags [SFPUAR]/[SFPUAR]". The order of the flags is not important. The first part specifies which flags can be set. The second part indicates a flag mask. Flags not included in the mask are ignored. Omitting the mask is equivalent to a mask of "SFPUAR". The meaning of each flag is presented below:

S	SYN
F	FIN
P	PSH or push
U	URG or urgent
A	ACK or acknowledgment
R	RST or reset

The second of the new options is the "keep state" option. "keep state" causes PF to place an entry into its state table for this packet. The state table will match any additional packets pertaining to this connection. It is processed before the ruleset. In addition to any convenience it adds, the state table also helps to protect against TCP session hijacking. The state table monitors fields in the packets that are not available to the ruleset to do some basic sanity checking. Because we have chosen to "keep state" on rule #1, the previous rule #2 is no longer needed.

Macros and other shortcuts. Macros are essentially variables. They allow us to specify a piece of information once and use it over and over again. If we need to change that information, it only needs to be changed in one place. How can we use this to simplify our rules. Suppose that our web server also ran a DNS server and that we were to allow zone transfers to external DNS servers. We might have a ruleset like this one:

MYSERVER = "FEC0::1"
NS1 = "FEC1::1"
NS2 = "FEC2::1"
pass in quick on gif0 inet6 proto tcp from any to $MYSERVER port = 80 flags S keep state
pass in quick on gif0 inet6 proto tcp from { $NS1, $NS2 } to $MYSERVER port = 53 flags S keep state
pass in quick on gif0 inet6 proto udp from any to $MYSERVER port = 53 keep state
pass in quick on dc1 all
pass out quick on dc1 all
block in all
block out all
block return-rst in proto tcp all
block return-rst out proto tcp all
block return-icmp in inet proto udp all
block return-icmp out inet proto udp all

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

block return-icmp6 in inet6 proto udp all
block return-icmp6 out inet6 proto udp all

We have added the use of macros so that we only have to enter commonly used addresses once. If ns1 changes its IP, it would be easy to update our firewall rules to reflect this change.

We have also added the use of curly braces ({}). This allows us to add a rule that will allow TCP connections to $MYSERVER on port 53 if they come from $NS1 *or* $NS2. We could also use curly braces in the protocol or several other fields.

Logging. Sometimes, we may want to log packets that match a firewall rule. This can be done with the "log" keyword. Let's modify our rules to log all blocked packets:

MYSERVER = "FEC0::1"
NS1 = "FEC1::1"
NS2 = "FEC2::1"
pass in quick on gif0 inet6 proto tcp from any to $MYSERVER port = 80 flags S keep state
pass in quick on gif0 inet6 proto tcp from { $NS1, $NS2 } to $MYSERVER port = 53 flags S keep state
pass in quick on gif0 inet6 proto udp from any to $MYSERVER port = 53 keep state
pass in quick on dc1 all
pass out quick on dc1 all
block in log all
block out log all
block return-rst in log proto tcp all
block return-rst out log proto tcp all
block return-icmp in inet log proto udp all
block return-icmp out inet log proto udp all
block return-icmp6 in inet6 log proto udp all
block return-icmp6 out inet6 log proto udp all

Now, we have instructed PF to create a log of the blocked packets. But, how is this logged? PF logs packets by sending them to the pflog0 interface. It is then up to pflogd(8) to capture the data and log it to a file.

Additional options. PF has several more options that we will not be using here. I encourage the reader to review the various documentation on PF and IPF included in the "References" section of this document for further information on these features.

Configuring our firewall

In this section, I will be discussing the ruleset I have implemented on my firewall. I will describe each section and explain its usefulness. We'll start by defining some macros:

V4IF = "dc0"
V4IP = "216.27.161.11"

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

```
V4TRUSTEDNET = "216.27.161.114/31"
V4TUNNELREMOTE = "193.113.58.80"
EXTIF = "gif0"
EXTIP = "2001:618:400::1:b037:4217"
TUNNELREMOTE = "2001:618:400::1:b037:4216"
INTIF = "dc1"
INTIP = "2001:618:400:465:203:6dff:fe00:3c64"
INTNET = "2001:618:400:465::/64"
INTBCAST = "2001:618:400:465:FFFF:FFFF:FFFF:FFFF"
INTROUTER = "2001:618:400:465::"
MAIL = "2001:618:400:465:290:27ff:fe3a:a4a9"
BACKUPMX = "2001:618:400:465:290:27ff:fe3a:b40d"
WEB = "2001:618:400:465:290:27ff:fe3a:a4a9"
FTP = "2001:618:400:465:290:27ff:fe3a:a4a9"
```

I have defined macros for the servers on my network, the network address, the broadcast address, the internal and external interfaces, the host's IP addresses, and the tunnel remote endpoint addresses. You will want to change this to reflect the information provided to you by your ISP and tunnel broker, your interface names, and the addresses of servers on your network.

```
#
# Default Rules
#

block in log all
block out log all
block return-rst in log proto tcp all
block return-rst out log proto tcp all
block return-icmp in log inet proto udp all
block return-icmp out log inet proto udp all
block return-icmp6 in log inet6 proto udp all
block return-icmp6 out log inet6 proto udp all
```

Next, we will define rules to block traffic and return appropriate error messages. Once again, we are covering both IPv4 and IPv6 here because the firewall itself has both versions configured. Since these rules do not include the "quick" option, rule processing will continue after evaluating these rules. In effect, we are establishing a "default deny" policy.

```
#
# Loopback
#
pass in quick on lo0 all
pass out quick on lo0 all
```

We're just talking to ourselves on lo0, so we will allow everything. It is presumed that a user of the machine will not want to purposefully generate harmful packets on lo0.

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

```
# ----------
# IPv4 Rules
# ----------

# Allow IPv6 tunnel
pass in quick on $V4IF inet proto ipv6 from $V4TUNNELREMOTE to $V4IP
pass out quick on $V4IF inet proto ipv6 from $V4IP to $V4TUNNELREMOTE

# Allow trusted hosts
pass in quick on $V4IF inet proto ip from $V4TRUSTEDNET to $V4IP
pass out quick on $V4IF inet proto ip from $V4IP to $V4TRUSTEDNET

# Allow outbound communication
pass out quick on $V4IF inet proto tcp from $V4IP to any flags S keep state
pass out quick on $V4IF inet proto udp from $V4IP to any keep state
pass out quick on $V4IF inet proto icmp from $V4IP to any keep state
```

This is the extent of our dealing with IPv4 in this firewall. The first set of rules will allow the encapsulated traffic created by the use of the gif interface. We are restricting such traffic so that it is only allowed to and from the remote tunnel IP. The second set of rules will allow traffic to and from our trusted network. This will allow us to administer the machine via IPv4 if we cannot do so over IPv6 for some reason. The final block is to allow outbound traffic. to the public Internet.

```
# ----------
# IPv6 Rules
# ----------

# Deny malicious packets
block in log quick inet6 proto tcp all flags FS/FS
block in log quick inet6 proto tcp all flags FSRPAU
block in log quick inet6 proto tcp all flags /FSRPAU
block in log quick inet6 proto tcp all flags FUP
```

This set of rules is designed to block packets with commonly used illegal flag combinations. These flag combinations are used to determine the OS used on a particular host or to bypass firewalls. Even though we are adopting a “default deny” policy, it is often good practice to explicitly block malicious traffic in case a mistyped rule otherwise opens the network a little more than intended.

```
#
# Blocked Ports
#

# Port 0, unused
```


© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

```
block in log quick on $EXTIF inet6 proto { tcp, udp } from any to any port = 0
# Echo
block in log quick on $EXTIF inet6 proto { tcp, udp } from any to any port = 7
# Chargen
block in log quick on $EXTIF inet6 proto { tcp, udp } from any to any port = 19
# TFTP
block in log quick on $EXTIF inet6 proto udp from any to any port = 69
# Portmapper
block in log quick on $EXTIF inet6 proto { tcp, udp } from any to any port = 111
# Microsoft loc-serv MS RPC end-point mapper
block in log quick on $EXTIF inet6 proto { tcp, udp } from any to any port = 135
# NetBIOS
block in log quick on $EXTIF inet6 proto { tcp, udp } from any to any port 136 <> 139
# SNMP
block in log quick on $EXTIF inet6 proto { tcp, udp } from any to any port = 161
block in log quick on $EXTIF inet6 proto { tcp, udp } from any to any port = 162
# xdmcp
block in log quick on $EXTIF inet6 proto { tcp, udp } from any to any port = 177
# More SNMP
block in log quick on $EXTIF inet6 proto { tcp, udp } from any to any port = 199
block in log quick on $EXTIF inet6 proto { tcp, udp } from any to any port = 391
# Win2k/XP SMB
block in log quick on $EXTIF inet6 proto { tcp, udp } from any to any port = 445
# pcserver backdoor
block in log quick on $EXTIF inet6 proto { tcp, udp } from any to any port = 600
# Linux mountd
block in log quick on $EXTIF inet6 proto { tcp, udp } from any to any port = 635
# AppleShare
block in log quick on $EXTIF inet6 proto { tcp, udp } from any to any port = 687
# Even more SNMP
block in log quick on $EXTIF inet6 proto tcp from any to any port = 705
# Sub-7
block in log quick on $EXTIF inet6 proto tcp from any to any port = 1243
# Millennium Worm backdoor
block in log quick on $EXTIF inet6 proto tcp from any to any port = 1338
# ingreslock backdoor
block in log quick on $EXTIF inet6 proto tcp from any to any port = 1524
# UPnP
block in log quick on $EXTIF inet6 proto udp from any to any port = 1900
# More SNMP
block in log quick on $EXTIF inet6 proto { tcp, udp } from any to any port = 1993
# NFSD
block in log quick on $EXTIF inet6 proto { tcp, udp } from any to any port = 2049
# DeepThroat
block in log quick on $EXTIF inet6 proto { tcp, udp } from any to any port = 2140
# More Sub-7
```


© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

```
block in log quick on $EXTIF inet6 proto tcp from any to any port = 27374
# More DeepThroat
block in log quick on $EXTIF inet6 proto tcp from any to any port = 3150
# More SNMP
block in log quick on $EXTIF inet6 proto tcp from any to any port = 4557
block in log quick on $EXTIF inet6 proto tcp from any to any port = 4559
# X-Windows, :0.0-:10.0
block in log quick on $EXTIF inet6 proto tcp from any to any port 6000 <> 6010
# More Sub-7
block in log quick on $EXTIF inet6 proto tcp from any to any port = 6776
# Back Orifice
block in log quick on $EXTIF inet6 proto udp from any to any port = 31337
```

Next, I'm explicitly blocking several ports. These are ports used by trojans, back doors, AppleShare, and Microsoft File and Print sharing and authentication mechanisms. In general, we don't want this traffic going in or out. You will notice there are no "block out" rules here. This is because I have not restricted the rules to any particular interface. Instead of blocking packets destined for port 31337 from going out gif0, they will be blocked coming in dc1. I have included comments above each block of rules showing exactly what they are blocking. If you have any services running on hosts behind the firewall that you do not want accessible from the outside world, consider adding them here as well.

```
#
# Trust the internal network
#

pass in quick on $INTIF inet6 from $INTNET to any
pass in quick on $INTIF inet6 from FE80::/10 to $INTIP
pass in quick on $INTIF inet6 from FE80::/10 to $INTBCAST
pass in quick on $INTIF inet6 from FE80::/10 to $INTROUTER
pass in quick on $INTIF inet6 from FE80::/10 to ::
pass in quick on $INTIF inet6 from FE80::/10 to FE80::/10
pass out quick on $INTIF inet6 from any to $INTNET
pass out quick on $INTIF inet6 from $INTIP to $INTNET
pass out quick on $INTIF inet6 from $INTROUTER to $INTNET
pass out quick on $INTIF inet6 from FE80::/10 to $INTNET
pass out quick on $INTIF inet6 from FE80::/10 to ::
pass out quick on $INTIF inet6 from FE80::/10 to FE80::/10

# Multicast
pass in quick on $INTIF inet6 from FE80::/10 to FF00::/8
pass in quick on $INTIF inet6 from FF00::/8 to FE80::/10
pass in quick on $INTIF inet6 from $INTNET to FF00::/8
pass in quick on $INTIF inet6 from FF00::/8 to $INTNET
pass in quick on $INTIF inet6 from :: to FF00::/8
pass in quick on $INTIF inet6 from FF00::/8 to ::
```


© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

pass out quick on $INTIF inet6 from FE80::/10 to FF00::/8
pass out quick on $INTIF inet6 from FF00::/8 to FE80::/10
pass out quick on $INTIF inet6 from $INTIP to FF00::/8
pass out quick on $INTIF inet6 from $INTROUTER to FF00::/8
pass out quick on $INTIF inet6 from FF00::/8 to $INTNET
pass out quick on $INTIF inet6 from :: to FF00::/8
pass out quick on $INTIF inet6 from FF00::/8 to ::

It is presumed that hosts on the internal network should be allowed to send whatever they want to the firewall's dc1 interface as long as it is from an address that should be present on the network. We will allow any traffic from our assigned netblock to anywhere. We are also allowing traffic from the link local addresses, FE80::/10. The multicast section is intended to allow traffic between multicast addresses, our firewall, and link local addresses. This traffic is necessary for router discovery and address auto-negotiation.

#
 # Block traffic from site local, broadcast, and invalid addresses
#

block in log quick on $EXTIF inet6 from FFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF to any
block in log quick on $EXTIF inet6 from :: to any
block in log quick on $EXTIF inet6 from FEC0::/112 to any
block in log quick inet6 from any to FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF
block in log quick inet6 from any to ::
block in log quick inet6 from any to FEC0::/112
block in log quick on $EXTIF inet6 from any to $INTNET
block in log quick on $EXTIF inet6 from any to $INTBCAST

These rules are intended to prevent smurf attacks and any other attacks involving the use of broadcast addresses. We do not want any traffic from the outside going to any of our broadcast addresses.

#
 # Trusted Hosts
#

pass in quick on gif0 inet6 proto ipv6-icmp from $TUNNELREMOTE to $EXTIP ipv6-icmp-type 128 keep state

Here we are specifying trusted hosts. These hosts will have certain additional privileges that the general public does not. Currently, we only have one trusted host. That is the remote tunnel endpoint. We are allowing it to ping us. Some tunnel brokers use an ICMPv6 ping to determine if the tunnel is active.

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

#
 # Protect internal net from spoofing
#

block in log quick on $EXTIF inet6 from $INTNET to any

This rule is added to block all packets coming in on the external interface that claim to be from our internal network. We do not want an attacker to be able to take advantage of one machine's trust of another machine on our internal network.

#
 # Allow outgoing connections
#

pass out quick on $EXTIF inet6 proto tcp from $INTNET to any flags S keep state
pass out quick on $EXTIF inet6 proto udp from $INTNET to any keep state
pass out quick on $EXTIF inet6 proto ipv6-icmp from $INTNET to any keep state
pass out quick on $EXTIF inet6 proto tcp from $EXTIP to any flags S keep state
pass out quick on $EXTIF inet6 proto udp from $EXTIP to any keep state
pass out quick on $EXTIF inet6 proto ipv6-icmp from $EXTIP to any keep state

These rules will allow both our firewall and hosts on our internal network to establish outgoing connections to the 6Bone. If you do not want any outgoing connections to the 6Bone, simply omit or comment these lines.

#
 # Allow incoming connections
#

 # SSH
pass in quick on $EXTIF inet6 proto tcp from any to $INTNET port = 22 flags S keep state

 # SMTP
pass in quick on $EXTIF inet6 proto tcp from any to { $MAIL, $BACKUPMX } port = 25 flags S keep state

 # POP3
pass in quick on $EXTIF inet6 proto tcp from any to $MAIL port = 110 flags S keep state

 # POP3s
pass in quick on $EXTIF inet6 proto tcp from any to $MAIL port = 995 flags S keep state

 # IMAP
pass in quick on $EXTIF inet6 proto tcp from any to $MAIL port = 143 flags S keep state

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

 # IMAPs
pass in quick on $EXTIF inet6 proto tcp from any to $MAIL port = 993 flags S keep state

 # HTTP
pass in quick on $EXTIF inet6 proto tcp from any to $WEB port = 80 flags S keep state

 # HTTPS
pass in quick on $EXTIF inet6 proto tcp from any to $WEB port = 443 flags S keep state

 # FTP
#pass in quick on $EXTIF inet6 proto tcp from any to $FTP port = 21 flags S keep state

This is our final set of rules. These rules will define which services the outside world is allowed to contact. Once again, there are comments above each rule block to explain which service that they allow. FTP has been left commented because the FTP server I am currently using does not support IPV6, yet. Also, allowing passive mode FTP may require additional rules.

Making it stick. Now, we are ready to apply the firewall. You will want to execute the next command from either the console or a serial terminal. Executing it from a network connection could lock you out of your firewall. To apply the firewall rules, execute "*pfctl -R /etc/pf.conf && pfctl –e*". If there are any errors, you will need to edit your /etc/pf.conf again and correct them before running the command again. Next, execute "*pflogd*".

You will not want to have to manually enable the firewall every time the system restarts. Edit /etc/rc.conf.local. Add the line "pf=YES". This will cause the firewall rules to be loaded and enabled on startup. It will also enable pflogd on startup.

One Final Step

At this point, we are done editing all of our configuration files. We have just one step left. To prevent the accidental modification of system files, we will remount certain filesystems as read-only, and add a few options to increase filesystem security. Edit /etc/fstab so that it looks something like this:

```
/dev/wd0a / ffs rw 1 1
/dev/wd0d /var ffs rw,nodev,noexec,nosuid 1 2
/dev/wd0e /usr ffs ro,nodev 1 2
/dev/wd1a /tmp ffs rw,nodev,noexec,nosuid 1 2
```

Your device names and mount points may differ. Let's go through these filesystems one by one:

- /var — The /var partition contains log files and some temp files. Therefore, it will need to have the read-write option. We also add "nodev", "noexec", and "nosuid". We do this because the filesystem contains no device files, executable files, or files or directories with the Set UID bit marked.

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

- /usr This filesystem is read-only. It only needs to be modified when upgrading the OS, recompiling the OS, or recompiling or adding additional applications. Also, we have added the "nodev" option because there are no device files on this partition. However, we do want executable files here and use of the Set UID bit is required in order for the OS to work correctly.
- /tmp This filesystem is mounted with the same options as /var. Though it does not store log files, there are temporary files here.

Now would be a good time to reboot. This will allow all remaining configuration changes and mount options to take effect. You will want to be at the console in case something goes wrong. If there is a typo in /etc/fstab, it will be necessary to enter single user mode to correct it.

Conclusion

A firewall should not be the only piece in your network security plan. It is just a start. Other items that should be included in your network security plan include various written guidelines and policies and an intrusion detection system. Modifications to the above procedure may also be necessary to comply with your organizations existing security and password policies. These topics are beyond the scope of this document, but I encourage readers who are truly interested in network security to research these topics on their own.

© SANS Institute 2002, Author retains full rights.

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

References

Acar, Can Erkin. “pflogd.” OpenBSD Manual Pages. 9 July 2001. URL: http://www.openbsd.org/cgi-bin/man.cgi?query=pflogd (25 Feb 2002).

“pf.conf.” OpenBSD Manual Pages. 8 July 2001. URL: http://www.openbsd.org/cgi-bin/man.cgi?query=pf.conf (25 Feb 2002).

Tran, Hoang Q. “OpenBSD firewall using pf.” 6 Dec 2001. URL: http://www.muine.org/~hoang/openpf.html (16 Feb 2002).

Brewer, Carl. “My 6bone router using OpenBSD.” URL: http://rollcage.bl.echidna.id.au/IPv6/openbsd.html (16 Feb 2002).

Brewer, Carl. “My 6bone firewall using OpenBSD.” URL: http://rollcage.bl.echidna.id.au/IPv6/openbsd-firewall.html (16 Feb 2002).

Paddon, Michael. “Joining the 6Bone.” Daemon News. Oct 2001. URL: http://www.daemonnews.org/200110/6bone.html (16 Feb 2002).

Coene, Wouter. “The OpenBSD Packet Filter HOWTO.” version 20011007. 7 Oct 2001. URL: http://www.inebriated.demon.nl/pf-howto/ (16 Feb 2002).

Conoboy, Brendan and Erick Fichtner. “The IPF HOW-TO.” 21 July 2001. URL: http://www.obfuscation.org/ipf/ipf-howto.html (16 Feb 2002).

Graham, Robert. “FAQ: Firewall Forensics (What am I seeing?).” version 0.4.1. 20 June 2000. http://www.robertgraham.com/pubs/firewall-seen.html (16 Feb 2002).

“OpenBSD.” version 1.368. 29 Jan 2002. URL: http://www.openbsd.org/ (26 Feb 2002).

“IPv6 News & Links.” 24 Feb 2002. URL: http://www.hs247.com (26 Feb 2002).

© SANS Institute 2002, Author retains full rights.

© SANS Institute 2002, As part of the Information Security Reading Room. Author retains full rights.

Last Updated: September 3rd, 2015

Upcoming SANS Training

Click Here for a full list of all Upcoming SANS Events by Location

SANS Crystal City 2015	Crystal City, VAUS	Sep 08, 2015 - Sep 13, 2015	Live Event
SANS Network Security 2015	Las Vegas, NVUS	Sep 12, 2015 - Sep 21, 2015	Live Event
SANS Seoul 2015	Seoul, KR	Sep 14, 2015 - Sep 19, 2015	Live Event
Data Breach Investigation Summit & Training	Dallas, TXUS	Sep 21, 2015 - Sep 26, 2015	Live Event
SANS Baltimore 2015	Baltimore, MDUS	Sep 21, 2015 - Sep 26, 2015	Live Event
SANS Perth 2015	Perth, AU	Sep 21, 2015 - Sep 26, 2015	Live Event
SANS Tallinn 2015	Tallinn, EE	Sep 21, 2015 - Sep 26, 2015	Live Event
SANS ICS Amsterdam 2015	Amsterdam, NL	Sep 22, 2015 - Sep 28, 2015	Live Event
SANS Bangalore 2015	Bangalore, IN	Sep 28, 2015 - Oct 17, 2015	Live Event
SANS DFIR Prague 2015	Prague, CZ	Oct 05, 2015 - Oct 17, 2015	Live Event
SANS Seattle 2015	Seattle, WAUS	Oct 05, 2015 - Oct 10, 2015	Live Event
SANS Tysons Corner 2015	Tysons Corner, VAUS	Oct 12, 2015 - Oct 17, 2015	Live Event
SOS: SANS October Singapore 2015	Singapore, SG	Oct 12, 2015 - Oct 24, 2015	Live Event
SANS Gulf Region 2015	Dubai, AE	Oct 17, 2015 - Oct 29, 2015	Live Event
SANS Cyber Defense San Diego 2015	San Diego, CAUS	Oct 19, 2015 - Oct 24, 2015	Live Event
SANS Tokyo Autumn 2015	Tokyo, JP	Oct 19, 2015 - Oct 31, 2015	Live Event
SANS South Florida 2015	Fort Lauderdale, FLUS	Nov 09, 2015 - Nov 14, 2015	Live Event
SANS Sydney 2015	Sydney, AU	Nov 09, 2015 - Nov 21, 2015	Live Event
SANS London 2015	London, GB	Nov 14, 2015 - Nov 23, 2015	Live Event
Pen Test Hackfest Summit & Training	Alexandria, VAUS	Nov 16, 2015 - Nov 23, 2015	Live Event
SANS Hyderabad 2015	Hyderabad, IN	Nov 24, 2015 - Dec 04, 2015	Live Event
SANS Cape Town 2015	Cape Town, ZA	Nov 30, 2015 - Dec 05, 2015	Live Event
SANS San Francisco 2015	San Francisco, CAUS	Nov 30, 2015 - Dec 05, 2015	Live Event
SANS Milan 2015	OnlineIT	Sep 07, 2015 - Sep 12, 2015	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced