Interested in learning more about security?

SANS Institute InfoSec Reading Room

Information Security

Linux Netfilter and FreeS/WAN are popular tools for creating low cost firewall and VPN gateways. Creating firewall rules for these systems can be relatively straight forward if you understand how they work. There are many commercial and free firewall tools that provide GUI interfaces for creating firewall rules and VPN connections. These tools allow for less understanding of how the firewall works since the management tool is trusted to create the correct rules. People that manually configure or troubleshooting problem...

Case Study of IPTables and Freeswan IPSEC
GIAC Security Essentials Certification (GSEC)
Practical Assignment Version 1.4b
Option 1 - Case Study in Information Security
Author: Eric Rupprecht
Date: 10 November 2003

Abstract

Linux Netfilter and FreeS/WAN are popular tools for creating low cost firewall and VPN gateways. Creating firewall rules for these systems can be relatively straight forward if you understand how they work. There are many commercial and free firewall tools that provide GUI interfaces for creating firewall rules and VPN connections. These tools allow for less understanding of how the firewall works since the management tool is trusted to create the correct rules. People that manually configure or troubleshooting problems need an understanding how the Netfilter and FreeS/WAN work to correctly administer the gateways. This guide will discuss Netfilter and FreeS/WAN explaining how each tool works and how the two interact. The goal is to show how a packet will flow through these tools to provide a better understanding of these technologies and enabling the administrator to write firewall rules with fewer errors.

Introduction

A while back, I was asked to upgrade a series of firewalls to enable IPSEC gateway-to-gateway tunneling between two sites. The current firewalls were running linux and using ipchains to enforce the firewall rules. The decision was made to us FreeS/WAN for the VPN connection and use Netfilter for the firewall. I started researching Netfilter and FreeS/WAN to gain an understanding of exactly how they work and interact. The following is a summary of this research.

Netfilter

Netfilter is a fourth generation Linux packet filtering software. Also called iptables, netfilter was introduced with the Linux 2.4 kernel and was designed to replace both the ipchains and ipfwadm legacy tools. There were two main advantages netfilter provides; simplified packet flow and stateful inspection[1].

Packet Flow through iptables

Understanding how packets flow through iptables allows the administrator to correctly create the firewall rules. The good news is that Netfilter simplified the packet flow through the chains as compared to ipchains.. The new model only

processes the necessary chains based on the ip appresses on the packet. There are three main chains INPUT, OUTPUT, and FORWARD that process the security policy. There are two chains that perform NAT and one chain for mangling the packet. The following diagram shows the test network for the examples in this section.

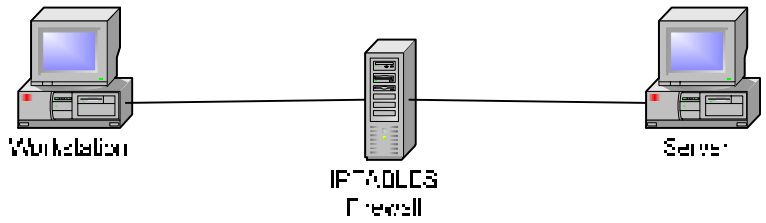

Figure 1: Iptables test network

The following diagram shows how a packet flows through the different iptables chains.

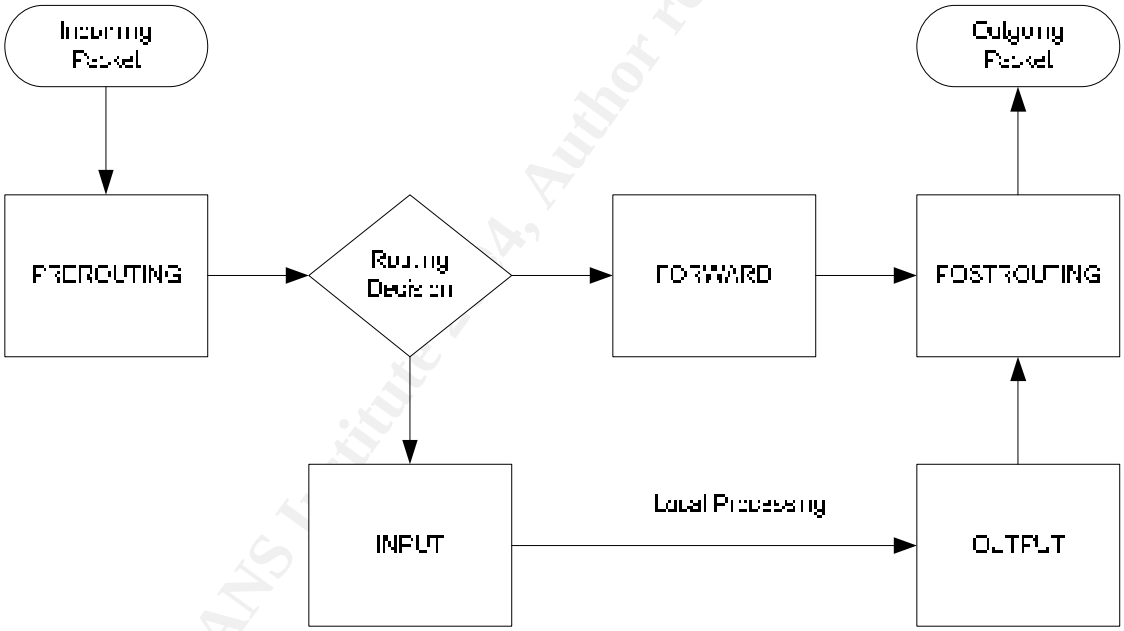

Figure 2: Iptables flow[1,2]

The flow through iptables is as follows:

1. The incoming packet is processed by the PREROUTING chain. This chain is used to NAT the destination on packets before any rules are applied[2]. The PREROUTING chain is frequently used to perform a one-to-one static destination NAT.
2. Next a routing decision is made based on the destination ip address of the packet. The packet is processed by the INPUT chain if the packet has a destination address that matches the IP address on the interface, i.e. the

packet is destine for this box[1]. A packet that is transferred to the INPUT chain will be compared to the rules attached to that chain to determine if it will be allowed. If the packet is allowed, it is transferred to the appropriate local process on the system. Replies to this packet will leave the local processing and traverse the OUTPUT chain.

3. A packet not destine for the box will be processed by the FORWARD chain. The rules in the chain are checked and, if accepted, the packet is routed to the correct interface. The packet will be dropped if ip forwarding is not enabled or does not know how to route the packet[1].
4. The OUTPUT chain is only traversed if the packets are initiated from the local box or are replies to packets that were destine for the box. Rules are checked and the packet passed out the correct interface[1].
5. The outgoing packet is processed by the POSTROUTING chain. This chain is used to NAT the source address on packets after rules are applied[2]. The POSTROUTING chain is frequently used to perform a hiding NAT.

Iptables Rules and Stateful Inspection

Stateful Inspection provides the ability to do connection tracking. In a non-stateful packet filter environment, a rule is created to allow the source address using a random source port to go to a destination on a service port. In addition, a rule is created to allow the return traffic. The return traffic rule needs to allow all ports above 1024 back to accommodate the randomly generated source port from the initial connection. This creates a very large hole in the firewall for the return traffic. Iptables solves this through connection tracking. Connection tracking keeps track of the packets that have been allowed. The return traffic can be compared against the connection table. Only the return traffic that matches the original source address and port, and destination address and port will be allowed. This closes the huge hole that had to be opened in non-stateful packet filters.

Connection tracking in iptables

In iptables, a rule is created to allow the traffic. The "- A INPUT" directive tells iptables to append the rule to the INPUT chain. The "-p" option will limit the rule to a specific protocol. This can be icmp, tcp, udp, or a protocol number like 50. The tcp and udp protocols will normally add the –dport option to specify the destination port. The "-m state –state NEW" option tells iptables to record information in the connection tracking table. This information can be used later to allow the return traffic. Finally, the "-j" option tells iptables to allow action to perform; accept, drop, or log. The following is an example of accepting a new icmp packet with state.

```
# iptables -A INPUT -p icmp -m state ! --state NEW -j ACCEPT
```

The return traffic can be automatically allowed by creating a rule that checks for established connections. The ”-m state –state ESTABLISHED” option tells the rule to look in the connection table to determine that the initiating connection has already been allowed and the return traffic should also be allowed. RELATED is also an option used here that is commonly needed for ftp. The following is an example of a generic rule allowing already established connections.

```
# iptables -A OUTPUT -m state ! --state ESTABLISHED, RELATED -j ACCEPT
```

Understanding packet flow is critical to correctly writing these rules. The following section will go through how packets flow through the chains.

Packet flow of stateful connections not destine for the firewall

Both of the state commands above would exist on the FORWARD chain when the packets are traversing the firewall. The initial packet is not destine for the firewall and the routing decision has iptables evaluate the packet in the FORWARD chain (shown in Figure 2). The return traffic is also not destine for the firewall and is also evaluated by the FORWARD chain.

```
# iptables -A FORWARD -p icmp -m state ! --state NEW -j ACCEPT
# iptables -A FORWARD -m state ! --state ESTABLISHED, RELATED -j ACCEPT
```

For the first test, the firewall in the test environment was set up to accept and log all packets.

```
#Set default policy to accept everything
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT
#Lets log everything on all chains
iptables -A INPUT -j LOG --log-prefix "INPUT"
iptables -A OUTPUT -j LOG --log-prefix "OUTPUT"
iptables -A FORWARD -j LOG --log-prefix "FORWARD"
```

A ping from the workstation to the server was issued. The Netfilter logs show only the forward chain being processed for both the echo request (icmp type 8 code 0) and the echo reply (icmp type 0 code 0).

```
Nov  2 08:32:35 gateway1 kernel: FORWARD IN=eth1 OUT=eth0 SRC=10.1.1.2
DST=10.3.3.2 LEN=60 TOS=0x00 PREC=0x00 TTL=127 ID=5277 PROTO=ICMP
TYPE=8 CODE=0 ID=512 SEQ=768
Nov  2 08:32:35 gateway1 kernel: FORWARD IN=eth0 OUT=eth1 SRC=10.3.3.2
DST=10.1.1.2 LEN=60 TOS=0x00 PREC=0x00 TTL=126 ID=18432 PROTO=ICMP
TYPE=0 CODE=0 ID=512 SEQ=768
```

Packet traces show the following packet flow across the firewall interfaces. Interface eth1 faces the workstation and eth0 faces the server.

```
eth1:  08:38:24.028043 10.1.1.2 > 10.3.3.2: icmp: echo request
```

```
eth0:  08:38:24.028456 10.1.1.2 > 10.3.3.2: icmp: echo request
eth0:  08:38:24.029874 10.3.3.2 > 10.1.1.2: icmp: echo reply
eth1:  08:38:24.030175 10.3.3.2 > 10.1.1.2: icmp: echo reply
```

This test verifies that packets being routed across the firewall only traverse the forward chain.

In the next test, the firewall was set up with a default policy to deny all packets. Then rules were added to allow all traffic and use stateful inspection.

```
#Set default policy to deny everything
iptables -P FORWARD DROP

# establish state and log everything
iptables -A FORWARD -m state --state NEW -j LOG --log-prefix "FORWARD
ACCEPT STATE NEW "
iptables -A FORWARD -m state --state NEW -j ACCEPT

iptables -A FORWARD -m state --state ESTABLISHED -j LOG --log-prefix
"FORWARD ACCEPT STATE EST "
iptables -A FORWARD -m state --state ESTABLISHED -j ACCEPT
```

A ping from the workstation to the server was issued. The net filter logs showed that the "-state NEW" rule was used for the icmp echo request, and the "--state ESTABLISHED" rule for the echo reply

```
Nov  2 16:09:01 gateway1 kernel: FORWARD ACCEPT STATE NEW IN=eth1
OUT=eth0 SRC=10.1.1.2 DST=10.3.3.2 LEN=60 TOS=0x00 PREC=0x00 TTL=127
ID=15772 PROTO=ICMP TYPE=8 CODE=0 ID=512 SEQ=1792
Nov  2 16:09:01 gateway1 kernel: FORWARD ACCEPT STATE EST IN=eth0
OUT=eth1 SRC=10.3.3.2 DST=10.1.1.2 LEN=60 TOS=0x00 PREC=0x00 TTL=126
ID=19456 PROTO=ICMP TYPE=0 CODE=0 ID=512 SEQ=1792
```

We can see above that the echo reply is handled by state, but how does iptables keep track of this. The file /proc/net/ip_conntrack tracks the state of the connections. The "-state NEW" option tells iptables to make an entry in the ip_conntrack table for the network connection. The "-state ESTABLISHED" option tells iptables to look in the ip_conntrack table and see if this is an already recorded (or allowed) connection. It is difficult to see the echo request state entry in the ip_conntrack table since it is only there a short time. As soon as the echo reply is processed, the connection is considered closed and the state entry removed. We can see the state entry for icmp by pinging a non-existent server so that the echo reply is not received.

```
icmp     1 26 src=10.1.1.2 dst=10.3.3.3 type=8 code=0 id=512 [UNREPLIED]
src=10.3.3.3 dst=10.1.1.2 type=0 code=0 id=512 use=1
```

Now that we have a good idea how state works, lets look at the case of TCP. For this test, the workstation telnets to the server on port 139 to establish a tcpip connection. The following trace shows that only the first packet is processed by

the "-state NEW" rule. All other packets in the connection are processed with state by the "-state ESTABLISHED" rule.

```
Nov  2 16:27:33 gateway1 kernel: FORWARD ACCEPT STATE NEW IN=eth1
OUT=eth0 SRC=10.1.1.2 DST=10.3.3.2 LEN=48 TOS=0x00 PREC=0x00 TTL=127
ID=17292 DF PROTO=TCP SPT=2337 DPT=139 WINDOW=64240 RES=0x00 SYN URGP=0
Nov  2 16:27:33 gateway1 kernel: FORWARD ACCEPT STATE EST IN=eth0
OUT=eth1 SRC=10.3.3.2 DST=10.1.1.2 LEN=48 TOS=0x00 PREC=0x00 TTL=126
ID=22784 DF PROTO=TCP SPT=139 DPT=2337 WINDOW=8760 RES=0x00 ACK SYN
URGP=0
Nov  2 16:27:33 gateway1 kernel: FORWARD ACCEPT STATE EST IN=eth1
OUT=eth0 SRC=10.1.1.2 DST=10.3.3.2 LEN=40 TOS=0x00 PREC=0x00 TTL=127
ID=17293 DF PROTO=TCP SPT=2337 DPT=139 WINDOW=64240 RES=0x00 ACK URGP=0
.
.
.
Nov  2 16:27:37 gateway1 kernel: FORWARD ACCEPT STATE EST IN=eth0
OUT=eth1 SRC=10.3.3.2 DST=10.1.1.2 LEN=40 TOS=0x00 PREC=0x00 TTL=126
ID=23296 DF PROTO=TCP SPT=139 DPT=2337 WINDOW=8759 RES=0x00 ACK URGP=0
```

This connection can be seen in the ip_conntrack table where it is shown as established.

```
tcp      6 431995 ESTABLISHED src=10.1.1.2 dst=10.3.3.2 sport=2354
dport=139 src=10.3.3.2 dst=10.1.1.2 sport=139 dport=2354 [ASSURED]
use=1
```

Packet flow of stateful connections destine for the firewall

The chains that are processed in this case are different. Refering fo Figure 2, you see that the packets flow through the INPUT and OUTPUT chains when they are destine for the firewall or are initiated by the firewall. The iptables state commands will need to be placed on both the INPUT and OUPUT chains. This is also true when iptables is used to secure a server with a single interface. Summarizing Figure 2 for this case, the initial packet is destine for the firewall and the routing decision has iptables evaluate the packet in the INPUT chain. The INPUT chain will have the state command establishing the NEW connection. The return traffic is coming from the firewall and will traverse the OUTPUT chain. The state command for the established traffic will be evaluated in the OUTPUT chain.

```
# iptables -A INPUT -p icmp -m state ! --state NEW -j ACCEPT
# iptables -A OUTPUT -m state ! --state ESTABLISHED, RELATED -j ACCEPT
```

For traffic initiating from the firewall to another server, the flow is reversed.

```
# iptables -A OUTPUT -p icmp -m state ! --state NEW -j ACCEPT
# iptables -A INPUT -m state ! --state ESTABLISHED, RELATED -j ACCEPT
```

We can demonstrate this in a series of tests. For the first test, the firewall was set up to accept and log all packets.

```
#Set default policy to accept everything
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT
#Lets log everything on all chains
iptables -A INPUT -j LOG --log-prefix "INPUT"
iptables -A OUTPUT -j LOG --log-prefix "OUTPUT"
iptables -A FORWARD -j LOG --log-prefix "FORWARD"
```

A ping was issued from the workstation to the firewall. The Netfilter logs show the echo request entering on the input chain and the echo reply leaving through the output chain.

```
Nov  2 16:33:01 gateway1 kernel: INPUT IN=eth0 OUT= SRC=10.1.1.2
DST=10.1.1.1 LEN=60 TOS=0x00 PREC=0x00 TTL=128 ID=17513 PROTO=ICMP
TYPE=8 CODE=0 ID=512 SEQ=2560
Nov  2 16:33:01 gateway1 kernel: OUTPUT IN= OUT=eth1 SRC=10.1.1.1
DST=10.1.1.2 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=48269 PROTO=ICMP
TYPE=0 CODE=0 ID=512 SEQ=2560
```

Packet traces show the icmp only on one interface.

```
eth1:  16:37:05.242283 10.1.1.1 > 10.1.1.2: icmp: echo reply
eth1:  16:37:05.241342 10.1.1.2 > 10.1.1.1: icmp: echo request
```

This test demonstrates that the forward chain is not traversed and only the input and output chains are used.

In the next test, the firewall was set up with a default policy to deny all packets. Then rules were added to allow all traffic and use stateful inspection.

```
#Set default policy to deny everything
iptables -P INPUT DROP
iptables -P OUTPUT DROP

# establish state
iptables -A INPUT -m state --state NEW -j LOG --log-prefix "INPUT
ACCEPT STATE NEW "
iptables -A INPUT -m state --state NEW -j ACCEPT
iptables -A OUTPUT -m state --state NEW -j LOG --log-prefix "OUTPUT
ACCEPT STATE NEW "
iptables -A OUTPUT -m state --state NEW -j ACCEPT

iptables -A INPUT -m state --state ESTABLISHED -j LOG --log-prefix
"INPUT ACCEPT STATE EST "
iptables -A INPUT -m state --state ESTABLISHED -j ACCEPT
iptables -A OUTPUT -m state --state ESTABLISHED -j LOG --log-prefix
"OUTPUT ACCEPT STATE EST "
iptables -A OUTPUT -m state --state ESTABLISHED -j ACCEPT
```

A ping from the workstation to the firewall was issued. The Netfilter logs showed that the "-state NEW" rule on the input chain was used for the icmp echo request, and the "--state ESTABLISHED" rule on the output chain was used for the echo reply.

```
Nov  2 16:46:29 gateway1 kernel: INPUT ACCEPT STATE NEW IN=eth1 OUT=
SRC=10.1.1.2 DST=10.1.1.1 LEN=60 TOS=0x00 PREC=0x00 TTL=128 ID=17822
PROTO=ICMP TYPE=8 CODE=0 ID=512 SEQ=4352
Nov  2 16:46:29 gateway1 kernel: OUTPUT ACCEPT STATE EST IN= OUT=eth1
SRC=10.1.1.1 DST=10.1.1.2 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=48276
PROTO=ICMP TYPE=0 CODE=0 ID=512 SEQ=4352
```

This case is different than the case of the forward chain since two chains are being processed. This means that the administrator needs to be careful to place the "--state NEW" and "--state ESTABLISHED" options on the correct chains.

The tcp protocol is a little more complicated. For this test, the workstation telnets to the firewall on port 111 to establish a tcpip connection. The following trace shows that only the first packet (the SYN packet) is processed by the "-state NEW" rule. All other packets in the connection are processed with state by the "-state ESTABLISHED" rule, but now we can see that the "-state ESTABLISHED" rules are being processed on both the input and output chains for a single connection.

```
Nov  2 16:53:09 gateway1 kernel: INPUT ACCEPT STATE NEW IN=eth1 OUT=
SRC=10.1.1.2 DST=10.1.1.1 LEN=48 TOS=0x00 PREC=0x00 TTL=128 ID=17880 DF
PROTO=TCP SPT=2383 DPT=111 WINDOW=64240 RES=0x00 SYN URGP=0
Nov  2 16:53:09 gateway1 kernel: OUTPUT ACCEPT STATE EST IN= OUT=eth1
SRC=10.1.1.1 DST=10.1.1.2 LEN=48 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF
PROTO=TCP SPT=111 DPT=2383 WINDOW=5840 RES=0x00 ACK SYN URGP=0
Nov  2 16:53:09 gateway1 kernel: INPUT ACCEPT STATE EST IN=eth1 OUT=
SRC=10.1.1.2 DST=10.1.1.1 LEN=40 TOS=0x00 PREC=0x00 TTL=128 ID=17881 DF
PROTO=TCP SPT=2383 DPT=111 WINDOW=64240 RES=0x00 ACK URGP=0
Nov  2 16:53:28 gateway1 kernel: INPUT ACCEPT STATE EST IN=eth1 OUT=
SRC=10.1.1.2 DST=10.1.1.1 LEN=41 TOS=0x00 PREC=0x00 TTL=128 ID=17884 DF
PROTO=TCP SPT=2383 DPT=111 WINDOW=64240 RES=0x00 ACK PSH URGP=0
Nov  2 16:53:28 gateway1 kernel: OUTPUT ACCEPT STATE EST IN= OUT=eth1
SRC=10.1.1.1 DST=10.1.1.2 LEN=40 TOS=0x00 PREC=0x00 TTL=64 ID=18256 DF
PROTO=TCP SPT=111 DPT=2383 WINDOW=5840 RES=0x00 ACK URGP=0
Nov  2 16:53:39 gateway1 kernel: INPUT ACCEPT STATE EST IN=eth1 OUT=
SRC=10.1.1.2 DST=10.1.1.1 LEN=40 TOS=0x00 PREC=0x00 TTL=128 ID=17886 DF
PROTO=TCP SPT=2383 DPT=111 WINDOW=64240 RES=0x00 ACK FIN URGP=0
Nov  2 16:53:39 gateway1 kernel: OUTPUT ACCEPT STATE EST IN= OUT=eth1
SRC=10.1.1.1 DST=10.1.1.2 LEN=40 TOS=0x00 PREC=0x00 TTL=64 ID=21063 DF
PROTO=TCP SPT=111 DPT=2383 WINDOW=5840 RES=0x00 ACK FIN URGP=0
Nov  2 16:53:39 gateway1 kernel: INPUT ACCEPT STATE EST IN=eth1 OUT=
SRC=10.1.1.2 DST=10.1.1.1 LEN=40 TOS=0x00 PREC=0x00 TTL=128 ID=17887 DF
PROTO=TCP SPT=2383 DPT=111 WINDOW=64240 RES=0x00 ACK URGP=0
```

This demonstrates that state is just another rule in iptables. The packet is processed by the chain that is appropriate based on the source and destination ipaddress on the packet. Both the input and output chains need “-state

ESTABLISHED" rules to allow the tcpip connection to be handled correctly. The ip_conntrack table entry can be seen in the tcp connection as follows.

```
tcp      6 431990 ESTABLISHED src=10.1.1.2 dst=10.1.1.1 sport=2386
dport=111 src=10.1.1.1 dst=10.1.1.2 sport=111 dport=2386 [ASSURED]
use=1
```

Iptables Limitations

Iptables is very powerful in securing IP traffic to the linux machine, but is limited in that it cannot secure ip traffic in transit on the network. Other mechanisms must be used to add this additional security. Application level encryption can be used to provide some security; PGP is used to encrypt email and SSL is typically used to encrypt web traffic. IPSEC can be used to encrypt all IP traffic at the network level. The next section will discuss IPSEC and how it can be used to secure data in transit on the network.

FreeS/WAN

FreeS/WAN is an IPSEC implementation for Linux that gives the ability to provide encryption and authentication. IPSEC is one of several technologies for encrypting data on the internet. PGP, SSH, and SSL are examples of application level encryption technologies. IPSEC has an advantage over the application level technologies in that it can "protect a mixture of application protocols running over a complex combination of media[3]." IPSEC can also provide protection without user interaction.

FreeS/WAN typically provides gate-to-gate IPSEC encryption linking two trusted networks. This will basically make the two networks look like they are directly linked by a router even though in actuality, the internet is separating these networks. The two trusted networks have a secured connection since all network traffic in the IPSEC tunnel (on the internet) is encrypted. Also, the security provided is transparent to the end user since encryption happens automatically between trusted networks.

Packet Flow through FreeS/WAN

Understanding how packets flow through FreeS/WAN will be described below. This information will be needed in the last section when the FreeS/WAN and Netfilter flows are combined. A simple test system will be used to demonstrate the packet flow.

Figure 3:IPSEC test network

Encrypted IPSEC Packets Enter the Gateway

The following diagram illustrates how an incoming ESP encrypted IPSEC packet flows through FreeS/WAN and leaves the gateway decrypted to the trusted network. There is an assumption that the IKE key exchange has already been completed and a security association has been made.

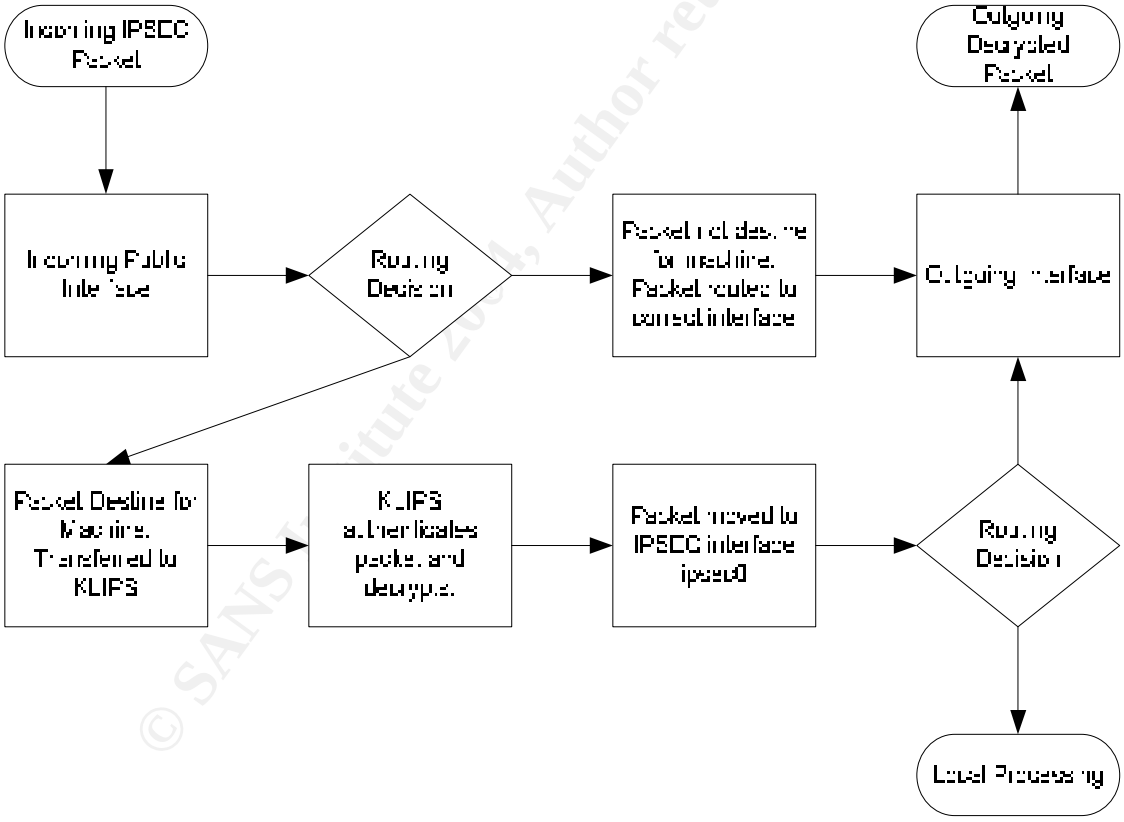

Figure 4: FreeS/WAN Incoming Encrypted Packet Flow[5]

The Packet flow through FreeS/WAN for IPSEC packets entering the gateway is as follows:

1. The incoming packet enters the gateway through the public interface. This packet is IPSEC encrypted (ESP protocol 50).
2. Next, a routing decision is made. If the ESP packet is not destine for the gateway, routing will transfer the packet to the correct interface and forward it on to the real destination. If the packet is destine for the gateway, It is transferred to the KLIPS process.
3. The KLIPS process authenticates the packet and decrypts the ESP packet to the original IP packet.
4. The unencrypted packet is moved to the ipsec network interface. This is specified in the ipsec configuration file and is normally the named ipsec0. A routing decision is now made based on the destination ip address of the unencrypted packet. The packet will either be transferred to the local system or routed to the correct network interface. In our example listed below, the unencrypted packet will be routed out to the network to the actual destination.

There are a couple of points here. The interface the IPSEC packets enter on only see encrypted packets. The ipsec interface ipsec0 will see the unencrypted ipsec packets enter the gateway.

Unencrypted Packets Enter the Gateway

Unencrypted packets that flow into the FreeS/WAN gateway follow the flow shown below.

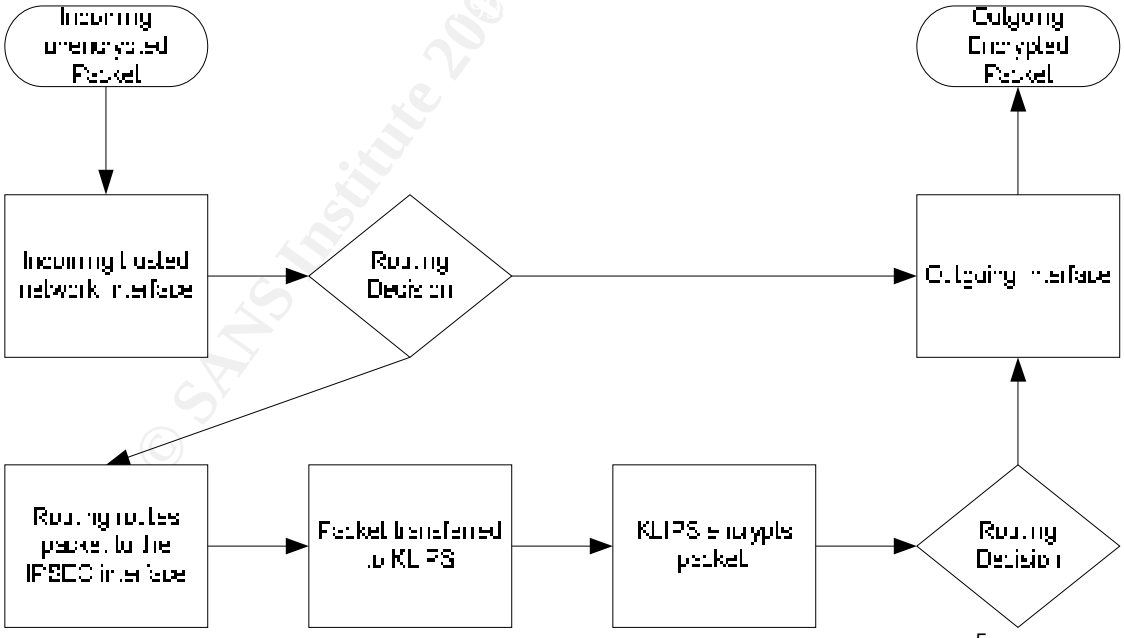

Figure 5: FreeS/WAN Incoming Unencrypted Packet Flow[5].

The Packet flow through FreeS/WAN for unencrypted packets entering the gateway is as follows:

1. The incoming unencrypted packet enters the gateway through the private interface.
2. Next, a routing decision is made. If the packet is not destine to a network defined as an ipsec encryption network, routing will transfer the packet to the correct interface and forward it on to the real destination. If the packet is destine to a network defined as an ipsec encryption network, the packet will be routed to the ipsec interface.
3. The unencrypted packet is moved to the ipsec network interface. The ipsec interface will transfer the packet to KLIPS.
4. The KLIPS process encrypts the packet.
5. Finally, a routing decision is made, and the encrypted packet is routed out to the appropriate FreeS/WAN gateway.

There are a couple of points here to consider about packets that are processed by FreeS/WAN. Packets enter the VPN gateway unencrypted on the private interface (eth1 in this example). These packets are also seen on the ipsec0 interface unencrypted. Packets leaving the gateway on the public interface (eth0) are encrypted and transmitted using ip protocol 50.

For out test, a ping was issued from the workstation to the server shown in Figure 3. The following trace was taken on the first gateway.

```
eth1:    17:24:16.834153 10.1.1.2 > 10.3.3.2: icmp: echo request
ipsec0:  17:24:16.834331 10.1.1.2 > 10.3.3.2: icmp: echo request
eth0:    17:24:16.835998 gateway1 > gateway2:
ESP(spi=0x8e54bd59,seq=0x126)

eth0:    17:24:16.840983 gateway2 > gateway1:
ESP(spi=0xdf4e5b06,seq=0x126)
ipsec0:  17:24:16.840983 10.3.3.2 > 10.1.1.2: icmp: echo reply
eth1:    17:24:16.841311 10.3.3.2 > 10.1.1.2: icmp: echo reply
```

This test demonstrated that the private interface (eth1) and ipsec interface (ipsec0) see unencrypted packets. The public interface (eth0) only sees encrypted packets.

FreeS/WAN Limitations

IPSEC does have some limitations. First, IPSEC cannot be secure if you computer or network is not secure. It is very important to secure network access points to your trusted network with a firewall. Secondly, IPSEC does not limit IP protocols that pass between gateways. FreeS/WAN is designed to pass all IP between the two trusted networks[3]. These FREES/WAN limitations and the limitation of iptables can be addressed by using both technologies together.

Using iptables and FreeS/WAN Together

Many of the limitation of FreeS/WAN and IPSEC can be eliminated be using each technology to secure the other. There are several firewall/VPN products available that combine these technologies out of the box. Smoothwall[4] is an example of a product that combines iptables and FreeS/WAN. Products like Smoothwall will give a GUI to configure rules and IPSEC, but an understanding of how iptables and IPSEC interact is still essential to properly securing and troubleshooting your network. This section is going to concentrate on manually configured iptables and FreeS/WAN to explain how the two products are used together.

IPSEC uses two protocols to send out encrypted packets onto the network. The first is Internet Key Exchange (IKE). IKE uses udp port 500 to setup the parameters for the encryption between the two vpn gateways. IKE is best known for exchanging encryption keys that will be used to encrypt the data. The second protocol used is Encapsulating Security Payload (ESP). This is where the encrypted data is transmitted using protocol 50.

Packet Flow through IPTABLES and FreeS/WAN

Packet flow through iptables and through FreeS/WAN was described in the previous sections. This section will show how packets flow through both technogies when used in combination. The test network will be the same as the one used in the FreeS/WAN packet flow section above. The following diagram shows an IPSEC encrypted packet entering the gateway, being decrypted, and routed to the trusted network.

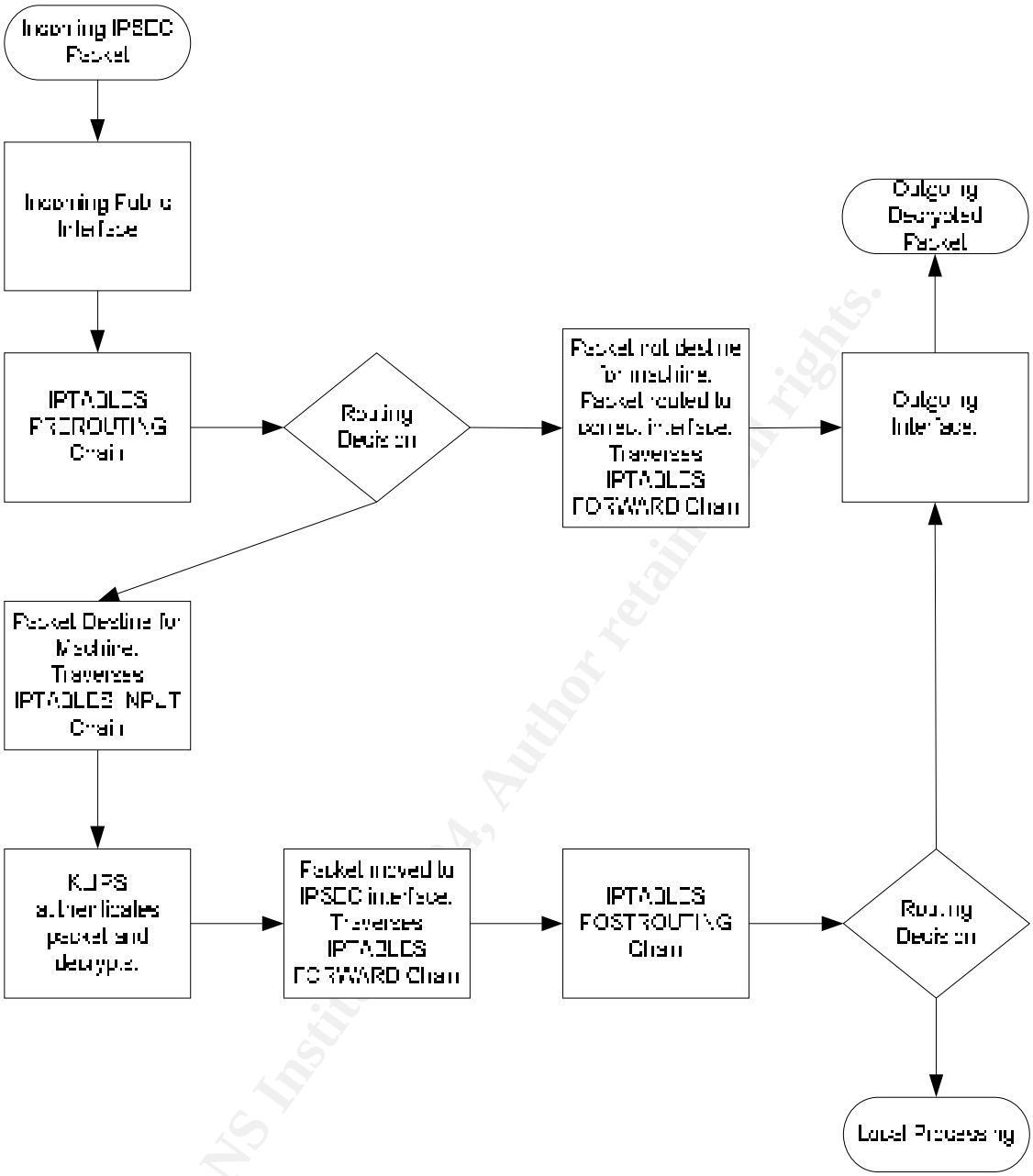

Figure 6: IPSEC packet entering Firewall-VPN Gateway

The Packet flow through Firewall-VPN for IPSEC packets entering the gateway is as follows:

1. The incoming IPSEC encrypted (ESP) packet enters the gateway through the public interface.
2. The incoming packet traverses the PREROUTING Chain. Typically, there will not be rules here that affect the ESP packet.
3. Next, a routing decision is made. If the ESP packet is not destine for the gateway, routing will transfer the packet to the correct interface and

forward it on to the real destination. If the packet is destine for the gateway, the packet traverses the INPUT chain.

4. If the ESP packet is allowed by the INPUT chain, it is transferred to the KLIPS process.
5. The KLIPS process authenticates and decrypts the ESP packet resulting in the original IP packet.
6. The unencrypted packet is moved to the ipsec network interface. Since our unencrypted packet has a source of the workstation and a destination of the server, the FORWARD chain is now traversed.
7. Next, the POSTROUTING chain is traversed. These rules would perform a source NAT if necessary.
8. Finally, a routing decision is now made based on the destination ip address of the unencrypted packet. The packet will be transferred to the correct network interface.

The ESP packets enter on the public interface (eth0) and are evaluated by the INPUT chain. Therefore, the INPUT chain will need to have rules to allow ESP (ip protocol 50) and the IKE key exchange (udp/500). When an unencrypted packet enters the gateway and is sent out IPSEC encrypted, the flow is reversed. This means that the output chain will also need rules to allow ESP and IKE.
The ipsec interface ipsec0 will always see unencrypted packets. Because of this, tcpdump can be run on this interface for debugging IPSEC network problems. Iptables rules can specify the ipsec0 interface to limit the traffic flowing through ipsec tunnels.

The following test will demonstrate the ipsec packet flowing through the combined system. For the test on the combined Netfilter and FreeS/WAN system, the gateways had iptables rules applied to accept all packets with state and log everything.

```
#Set default policy to deny everything
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
# establish state
iptables -A INPUT -m state --state NEW -j LOG --log-prefix "INPUT
ACCEPT STATE NEW "
iptables -A INPUT -m state --state NEW -j ACCEPT
iptables -A OUTPUT -m state --state NEW -j LOG --log-prefix "OUTPUT
ACCEPT STATE NEW "
iptables -A OUTPUT -m state --state NEW -j ACCEPT
iptables -A FORWARD -m state --state NEW -j LOG --log-prefix "FORWARD
ACCEPT STATE NEW "
iptables -A FORWARD -m state --state NEW -j ACCEPT

iptables -A INPUT -m state --state ESTABLISHED -j LOG --log-prefix
"INPUT ACCEPT STATE EST "
iptables -A INPUT -m state --state ESTABLISHED -j ACCEPT
iptables -A OUTPUT -m state --state ESTABLISHED -j LOG --log-prefix
"OUTPUT ACCEPT STATE EST "
```

```
iptables -A OUTPUT -m state --state ESTABLISHED -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED -j LOG --log-prefix
"FORWARD ACCEPT STATE EST "
iptables -A FORWARD -m state --state ESTABLISHED -j ACCEPT
```

The workstation sent a ping to the server. The following trace shows the packet traverse the interfaces of the first gateway.

```
eth1:    17:42:52.387349 10.1.1.2 > 10.3.3.2: icmp: echo request
ipsec0:  17:42:52.387814 10.1.1.2 > 10.3.3.2: icmp: echo request
eth0:  17:42:52.388474 gateway1 > gateway2:
ESP(spi=0x8e54bd59,seq=0x12f)
eth0:  17:42:52.392692 gateway2 > gateway1:
ESP(spi=0xdf4e5b06,seq=0x12f)
ipsec0:  17:42:52.392692 10.3.3.2 > 10.1.1.2: icmp: echo reply
eth1:  17:42:52.393750 10.3.3.2 > 10.1.1.2: icmp: echo reply
```

The iptables log entries show all three chains being traversed for the ping.

```
Nov  3 17:42:52 gateway1 kernel: FORWARD ACCEPT STATE NEW IN=eth1
OUT=ipsec0 SRC=10.1.1.2 DST=10.3.3.2 LEN=60 TOS=0x00 PREC=0x00 TTL=127
ID=45865 PROTO=ICMP TYPE=8 CODE=0 ID=512 SEQ=21505
Nov  3 17:42:52 gateway1 kernel: OUTPUT ACCEPT STATE EST IN= OUT=eth0
SRC=1.1.1.2 DST=1.3.3.2 LEN=112 TOS=0x00 PREC=0x00 TTL=64 ID=28614
PROTO=ESP SPI=0x8e54bd59
Nov  3 17:42:52 gateway1 kernel: INPUT ACCEPT STATE EST IN=eth0 OUT=
MAC=00:a0:c9:c5:4a:2a:00:20:af:ab:a7:0a:08:00 SRC=1.3.3.2 DST=1.1.1.2
LEN=112 TOS=0x00 PREC=0x00 TTL=63 ID=4997 PROTO=ESP SPI=0xdf4e5b06
Nov  3 17:42:52 gateway1 kernel: FORWARD ACCEPT STATE EST IN=ipsec0
OUT=eth1 SRC=10.3.3.2 DST=10.1.1.2 LEN=60 TOS=0x00 PREC=0x00 TTL=126
ID=39681 PROTO=ICMP TYPE=0 CODE=0 ID=512 SEQ=21505
```

This demonstrates the need for rules on all three interfaces. The INPUT and OUTPUT chains handle the encrypted traffic, and the FORWARD chain processes the unencrypted traffic.

Writing iptables rules to secure IPSEC

Writing IP tables rules for FreeS/WAN connections becomes relatively easy with the knowledge we gained above. We will walk through an example for the rules on the first gateway to secure all connections with the second gateway and only allow icmp through the IPSEC tunnel.
First, we will set the default policy to deny all packets.

```
#Set default policy to deny everything
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
```

Next, we will create rules to allow state for established connections on all chains.

```
iptables -A INPUT -m state --state ESTABLISHED -j ACCEPT
```

```
iptables -A OUTPUT -m state --state ESTABLISHED -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED -j ACCEPT
```

The following rules will allow the IKE and ESP requests on the public interface of the gateway. These need to be on both the input and output chains.

```
# IKE negotiations
iptables -A INPUT –i eth0 -p udp --sport 500 --dport 500 -m state --
state NEW -j ACCEPT
iptables -A OUTPUT –i eth0 -p udp --sport 500 --dport 500 -m state --
state NEW -j ACCEPT
# ESP encryption and authentication
iptables -A INPUT  –i eth0 -p 50 -j ACCEPT
iptables -A OUTPUT –i eth0 -p 50 -j ACCEPT
```

The last rules that we need are to allow the ICMP to flow through the IPSEC tunnel. These rules are placed on the forward chain and can be linked to the ipsec0 interface if desired.

```
# Allow icmp through the tunnel
iptables -A FORWARD –i ipsec0 –p icmp -m state --state NEW -j ACCEPT
```

This is all we need to secure the VPN gateway in our example. More forward rules can be added as needed to allow additional traffic through the ipsec tunnel.

Limitations of combined iptables and FreeS/WAN

The above system solved many limitations, but not all. The solution does not authenticate users. Authentication is based off of machines or networks. Another product would need to be added to our solution to provide user level authentication.

Conclusion

The knowledge gain above was applied to a mesh VPN network to link four remote sites together. Creating Netfilter rules was straight forward and we have high confidence that the rules put in place are properly securing the networks. This guide should provide you with enough knowledge to write secure rules for VPN networks.

References

1. Russell, Rusty. "Linux 2.4 Packet Filtering HOWTO" 24 Jan 2002. URL: http://www.netfilter.org/documentation/HOWTO//packet-filtering-HOWTO.txt (6 Oct. 2003).
2. Russell, Rusty. " Linux 2.4 NAT HOWTO" 14 Jan 2002, Version 1.18. URL: http://www.netfilter.org/documentation/HOWTO/NAT-HOWTO.txt (7 Oct. 2003).

3. Unknown. “FreeS/WAN documentation" 15 April 2003. URL: http://www.freeswan.org/freeswan_trees/freeswan-2.02/doc/ (7 Oct. 2003).
4. Unknown “SmoothWall: Secure your digital world”, URL: http://www.smoothwall.org/about/ (20 Oct. 2003).
5. Friede, Marcus “linux-ipsec: diagram to show packet flow *DONE*” 01 Dec 2000. URL: http://www.sandelman.ottawa.on.ca/linux-ipsec/html/2000/12/msg00006.html (20 Oct. 2003).
6. Kent, S. “IP Encapsulating Security Payload (ESP)” July 2003. URL: http://www.ietf.org/internet-drafts/draft-ietf-ipsec-esp-v3-06.txt (3 Nov 2003).
7. Kaufman, Charlie “Internet Key Exchange (IKEv2) Protocol” Oct 9, 2003 URL: http://www.ietf.org/internet-drafts/draft-ietf-ipsec-ikev2-11.txt (3 Nov 2003)
8. Insolvivile, Gianluca “Kernel Korner: Inside the Linux Packet Filter” February 2002, URL: http://www.linuxjournal.com/article.php?sid=4852 (4 Nov 2003)

Last Updated: March 18th, 2018

Upcoming SANS Training

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

SANS Munich March 2018	Munich, DE	Mar 19, 2018 - Mar 24, 2018	Live Event
SANS Pen Test Austin 2018	Austin, TXUS	Mar 19, 2018 - Mar 24, 2018	Live Event
SANS Secure Canberra 2018	Canberra, AU	Mar 19, 2018 - Mar 24, 2018	Live Event
SANS Boston Spring 2018	Boston, MAUS	Mar 25, 2018 - Mar 30, 2018	Live Event
SANS 2018	Orlando, FLUS	Apr 03, 2018 - Apr 10, 2018	Live Event
SANS Abu Dhabi 2018	Abu Dhabi, AE	Apr 07, 2018 - Apr 12, 2018	Live Event
Pre-RSA® Conference Training	San Francisco, CAUS	Apr 11, 2018 - Apr 16, 2018	Live Event
SANS Zurich 2018	Zurich, CH	Apr 16, 2018 - Apr 21, 2018	Live Event
SANS London April 2018	London, GB	Apr 16, 2018 - Apr 21, 2018	Live Event
SANS Baltimore Spring 2018	Baltimore, MDUS	Apr 21, 2018 - Apr 28, 2018	Live Event
Blue Team Summit & Training 2018	Louisville, KYUS	Apr 23, 2018 - Apr 30, 2018	Live Event
SANS Seattle Spring 2018	Seattle, WAUS	Apr 23, 2018 - Apr 28, 2018	Live Event
SANS Riyadh April 2018	Riyadh, SA	Apr 28, 2018 - May 03, 2018	Live Event
SANS Doha 2018	Doha, QA	Apr 28, 2018 - May 03, 2018	Live Event
SANS SEC460: Enterprise Threat Beta Two	Crystal City, VAUS	Apr 30, 2018 - May 05, 2018	Live Event
Automotive Cybersecurity Summit & Training 2018	Chicago, ILUS	May 01, 2018 - May 08, 2018	Live Event
SANS SEC504 in Thai 2018	Bangkok, TH	May 07, 2018 - May 12, 2018	Live Event
SANS Security West 2018	San Diego, CAUS	May 11, 2018 - May 18, 2018	Live Event
SANS Melbourne 2018	Melbourne, AU	May 14, 2018 - May 26, 2018	Live Event
SANS Northern VA Reston Spring 2018	Reston, VAUS	May 20, 2018 - May 25, 2018	Live Event
SANS Amsterdam May 2018	Amsterdam, NL	May 28, 2018 - Jun 02, 2018	Live Event
SANS Atlanta 2018	Atlanta, GAUS	May 29, 2018 - Jun 03, 2018	Live Event
SANS Rocky Mountain 2018	Denver, COUS	Jun 04, 2018 - Jun 09, 2018	Live Event
SANS London June 2018	London, GB	Jun 04, 2018 - Jun 12, 2018	Live Event
DFIR Summit & Training 2018	Austin, TXUS	Jun 07, 2018 - Jun 14, 2018	Live Event
SANS Milan June 2018	Milan, IT	Jun 11, 2018 - Jun 16, 2018	Live Event
SEC487: Open-Source Intel Beta One	OnlineVAUS	Mar 19, 2018 - Mar 24, 2018	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced