# Global Information Assurance Certification Paper

**Mario Serrano**
**GCFW Version 1.6**
**GIAC Enterprises Fortunes**

**I - Security Architecture**

GIAC Enterprises deals in the online sale of fortune cookie sayings. It has:

- Customers that purchase bulk online fortunes.
- Suppliers that supply fortunes.
- International partners that translate and resell fortunes.

**Customers** will connect to the web-based online ordering system and place their orders. Encryption will be required (that means HTTPS). Once an order has been validated, the paper fortune cookie sayings will be shipped.

**Suppliers** are connected to, in order to ask for more popular sayings, according to inventory levels. They have no special access to GIAC Enterprises' systems. All connections are initiated from GIAC, and are done using their encrypted web-based ordering system, where GIAC has an account.

International **partners** browse through the cookie sayings catalog on GIAC's web server. They place orders as clients do, so that they can be shipped materials to, for reselling purposes. They also have a web-based application available to them, that can be used to specify translated phrases, and the amount of paper cookie sayings to be shipped to them. Their accounts have to be preprogrammed into GIAC's systems, and their access must come, network-wise, from the VPN connection (validated). GIAC doesn't like getting stuck with unsold foreign language inventory, so translated material is contracted to suppliers and then shipped entirely to partners.

The security architecture reflects these facts. It includes several devices to implement it. There is a filtering router, an external firewall, a VPN box to connect to the business partners and internal firewalls.

**Filtering router**

We chose a Cisco 3640. It has good performance, and supports technologies such as Access Control Lists, Commited Access Rate (CAR). Its operating system, IOS, is well known. We will just make sure that we have a fairly recent version, since there might be denial of service problems with older ones.

It is placed right at the end of the link that connects GIAC to its ISP. It has one WAN interface, and two Fast Ethernet interfaces. An extra WAN interface can be added, since it is a modular router, in order to have redundancy in case of network problems.

The router has to have several things configured, besides Access Control Lists. Among them we can mention:

- Source routing support eliminated. This prevents several attacks. Can be easily done with a single command.
- No web configuration for the router. It is usually cause for problems such as denial-of-service attacks.
- "*No ip unreachables*" command set, to avoid sending extra information to hackers.
- Use of the *rate-limit* command to set permissible levels for SYN packets. This command, part of the

CAR (Committed Access Rate) package from Cisco could limit the damage from SYN Flood attacks.

- Small services (both UDP and TCP) must be disabled. This prevents usage of things like the echo service. The commands "*no service udp-small-servers*" and "*no service tcp-small-servers*" must be used for this purpose.
- Enable (high security level) secret defined in the configuration, instead of an enable password that can be decrypted easily.
- Interfaces should have no support for ip redirects and ip directed-broadcasts (the latter to avoid Smurf-type attacks).
- This device will do NAT for the private address networks used in several firewall interfaces.
- Classless configuration is enabled.
- Cisco command "ip subnet-zero" must be enabled too, since it allows us to use the IP addresses we want in this case.

One LAN connection is connected to the firewall. The other one to the VPN system.

**External firewall**

GIAC has knowledgeable Linux/UNIX staff. A RedHat Linux 7.1 (http://www.redhat.com/ )box has been configured to be its firewall. All unnecessary services have been eliminated, and all patches available until early October 2001 have been installed. Among the patches, there is a new version of the Linux Kernel: 2.4.5, that fixes several vulnerabilities.

IPChains is used. The advantage of the platform chosen is that if desired, the firewall can be configured using IPTables, a stateful filtering technology. It is not used in this project because we have not finished evaluating its stability. Nevertheless we have an upgrade path available.

Six NICs have been installed:

eth0: Internal Network interface
eth1: External network interface
eth2: DNS/Syslog interface
eth3: VPN interface
eth4: WWW interface
eth5: Database interface

The firewall is placed at the core of the system. It separates the networks, isolating them to prevent further security problems if systems are compromised. All traffic will pass through it.

This device is also the point where GIAC would consider improvements in its security architecture. Once the stateful IPTables technology is trusted enough, a good measure would be to substitute a single firewall configuration with a two firewall one. The firewall closest to the filtering router would be the fastest, an IPChains-based one; it would then pass connections to a second IPTables-based one, that would inherit all other connections.

**VPN**

Since GIAC is a Linux/Unix shop, it is installing a Linux solution. Free/SWAN is the standard in Linux for IPSec VPN. Unfortunately it requires Kernel patching and further configuration. Therefore a preconfigured, prepatched Linux distribution with Free/SWAN is used for simplicity's sake.

Smoothwall 0.99 (http://www.smoothwall.org ), a Linux distribution with Free/SWAN 1.9 support built in was chosen. It will be installed in a fast, two network-interface box just for this task.

Smoothwall allows for easy web-based VPN configuration, using shared secrets. It also supports PPTP. Under-the-hood configuration would be required if GIAC wishes to use RSA certificates.

The VPN box receives connections directly from the filtering router, but also passes decoded, unencrypted traffic to one interface of the firewall. We do not inject traffic directly to the internal network.

We have given the outer VPN box interface a public IP address. This will allow GIAC to use packet header authentication, since there is no NAT-related conversions taking place.

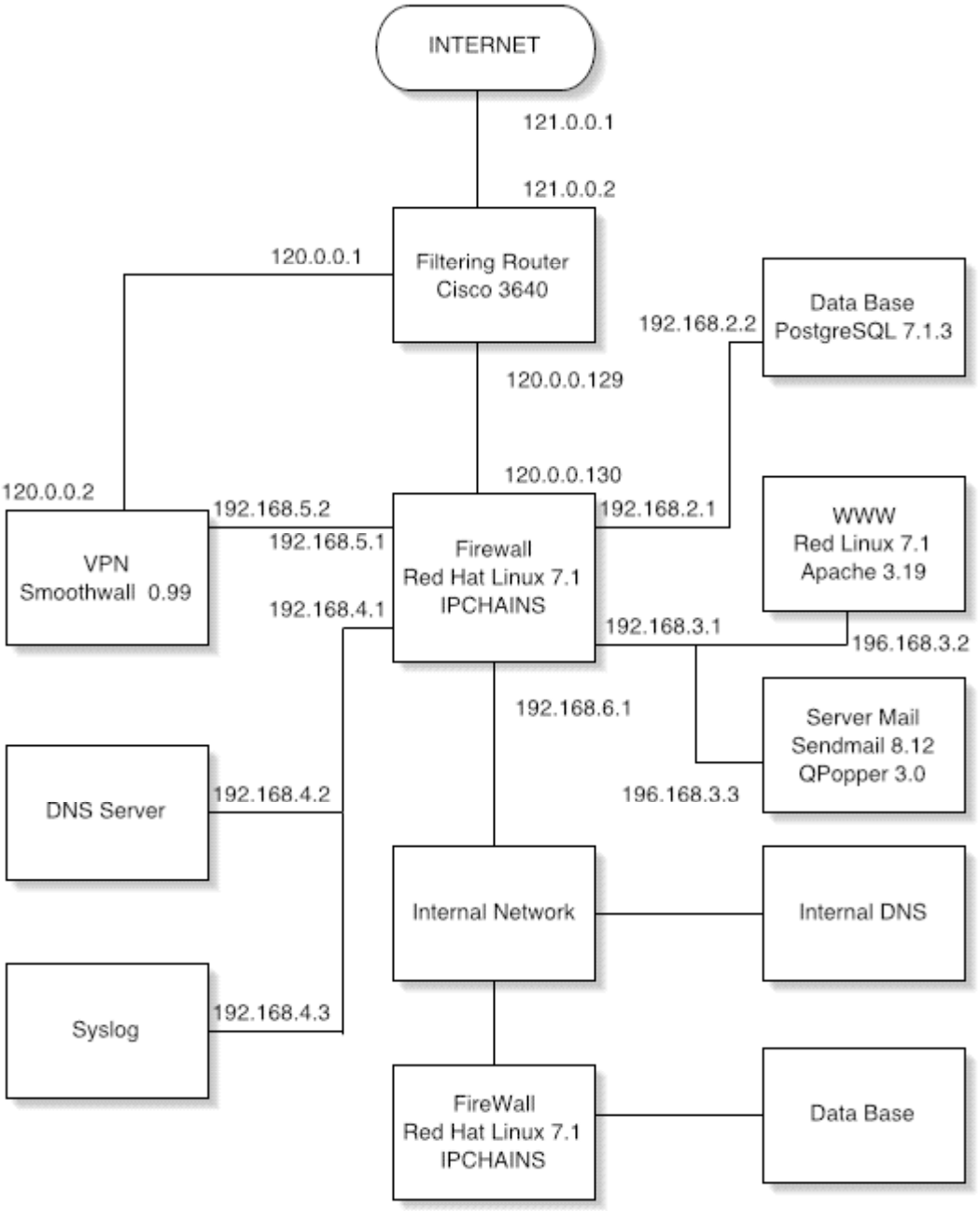
**Internal firewalls**

They use Linux, and IPChains, as mentioned above. They protect internal, important systems such as accounting databases from internal users.

**Intrusion Detection Systems**

Snort 1.81 (http://www.snort.org) is used as IDS. One system is placed at the WWW/Mail network, and another at the firewall's internal network interface.

The network layout is as follows:



## II - Security Policy

### Border router policy

It is a Cisco box, which can handle several protocols; IP is just one of them. It is possible to perform filtering using Cisco's Access Control Lists.

Access Control Lists, or ACLs are specified in the configuration mode, accessed with the *configuration terminal* command, given the permissions to do it and the usage of the enable-level access (accessed with the *enable* command).

ACLs can be thought as checklists that are used whenever a packet goes to or from an interface. Those checklists have to be unique. Only one can be used outbound or inbound. It is possible, though, to have two assigned to a given interface: one inbound and one outbound.

Cisco ACLs are written with the syntax

*access-list number action matching_conditions*

*number* has to be between 0 and 199 to specify rules used for the IP protocol. If it is between 0 and 99, we are referring to Standard IP ACLs, in which the matching conditions can only be specified as a function of the source address only (and for IP only!). If number is between 100 and 199 it is an Extended IP ACL, that can specify both source and destination addresses, with ports for both UDP and TCP. Since our rules are complex, we will use Extended IP ACLs.

*Action* can be either word, *permit* or *deny*. This would be the action that would be taken if the *matching_conditions* do that, match the packet. Any checklist or ACL is processed from the first line downward. Whenever a match is done, action is taken: *permit* allows the packet to go on to whichever interface it is destined to. *deny* would prevent a packet from reaching its destination.

Regarding *matching_conditions* for an Extended IP ACL, they are specified protocol first, source second, destination, third. A source or destination address can be specified using three parameters:

| | |
|---|---|
| IP address: | This is just the same old IP address written in the normal notation, like 192.168.0.0 |
| Cisco netmask: | This is where it is important to be careful! Instead of the "normal" netmask notation, in which binary "1" is used on each bit belonging to the network address, this is its complement. For example, the normal netmask for a class C network would be 255.255.255.0; the Cisco netmask would be 0.0.0.255 |
| Port specification: | Ports can be specified as direct matches ( *eq 80* ), ranges ( range *135 139* ), inequalities ( *gt 1024* , *lt 1024* - greather than, and less than, respectively). *Established* is a keyword that can specify that matching packets are the ones belonging to "established" TCP connections (those that do not have their SYN bit set). There are several modes, that will not be covered in this document. It is possible to look at all the available options by pressing "?" at the precise place in the command line. |

There are shortcuts for the IP address/Cisco netmask combination. Those are meant for two specific

cases: a single host and any host. A single host can be specified with *host IP* (for example, *host 192.168.0.1*). Any host can be specified with the keyword *any*.

There is one important tip regarding Cisco ACLs. There is an implicit last rule, a "deny everything ip from anywhere to anywhere" rule. If things do not work well, perhaps this has not been taken into account.

ACLs can be assigned for both incoming or outgoing packets. In the router configuration, the only thing needed would be to specify, FOR THE RESPECTIVE INTERFACE, *access-group number in* (incoming) or *access-group number out* (outgoing packets), where number is the access list number we have already configured.

Proper configuration would have us defining both inbound and outbound ACLs to be applied at the router's WAN interface. The commented Cisco ACL configuration goes as follows:

```
! Inbound ACL. Note that the ACL number is 140, indicating an extended IP ACL.
! Other important thing is that "!" indicates a comment line.
!
! First we deny any access to traffic from any private address coming through the WAN interface
! We need to block both TCP and UDP. The easy way is then to block the whole IP protocol.
! The addresses to block are 192.168.0.0/16, 10.0.0.0/8, and 172.16.0.0/12. Other important
! thing to note here is that the lines end up with "log". This means that any match will be
! logged. If a syslog server has been configured, the router will send the information to it.
!
! The next line is the one the router will try to match packets against first. Then all others go.
access-list 140 deny   ip 192.168.0.0 0.0.255.255 any log
access-list 140 deny   ip 10.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 172.16.0.0 0.15.255.255 any log
!
! Note that netmasks should be specified using Cisco's notation. Also, the "any" keyword
! has been used to indicate that any destination is to be considered, paired with each source.
!
! Denying IANA reserved address space to prevent spoof-based attacks.
! These addresses are not used anywhere, and could be used in spoofing attacks.
! Again, we block the whole IP protocol, blocking TCP, UDP and ICMP at the same time.
! any indicates "any destination"
access-list 140 deny   ip 1.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 2.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 5.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 7.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 23.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 27.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 31.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 37.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 39.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 41.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 42.0.0.0 0.255.255.255 any log
```

```
access-list 140 deny   ip 58.0.0.0 1.255.255.255 any log
access-list 140 deny   ip 60.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 63.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 64.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 65.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 66.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 67.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 68.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 69.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 70.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 71.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 72.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 73.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 74.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 75.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 76.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 77.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 78.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 79.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 80.0.0.0 15.255.255.255 any log
access-list 140 deny   ip 96.0.0.0 15.255.255.255 any log
! The next line also includes the loopback address, that should not be coming through the WAN
! interface, anyway.
access-list 140 deny   ip 112.0.0.0 15.255.255.255 any log
access-list 140 deny   ip 217.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 218.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 219.0.0.0 0.255.255.255 any log
access-list 140 deny   ip 220.0.0.0 63.255.255.255 any log
!
! We deny any class D Multicast address traffic, since we don't need anything like it
access-list 140 deny   ip 224.0.0.0 15.255.255.255 any log
!
! Deny class E traffic
access-list 140 deny   ip 240.0.0.0 7.255.255.255 any log
!
! An anti-spoofing rule.  Since we use the 120.0.0.0/24 network, we should not expect to see
! incoming WAN packets with this address, going anywhere (that's the reason behind the any
! keyword).  Hackers could use GIAC's same address space to
! fake traffic coming from a trusted IP address.
access-list 140 deny   ip 120.0.0.0 0.0.0.255 any log
!
! Blocking Microsoft protocols.  We'd better put these somewhere.  The range keyword is used to
! indicate that the following two numbers represent the initial port and the final port in a range to
! be used for matching purposes.
! Ports 135 through 139 are used in Microsoft Networking protocols for W9x and Win NT
! Failure to block them could, in a network, result in denial of service attacks against it, or even
! unwanted file sharing across the Internet.  They are meant to be used inside a LAN, anyway.
```

```
! Source is any, destination is any:  no matter the origin, no matter the destination, if the ports match,
! we want that traffic to be eliminated.
!
access-list 140 deny   tcp any any range 135 139 log
access-list 140 deny   udp any any range 135 139 log
!
! Port 445 is used from Windows 2000 forward for its networking.  Since it is a single port,
! we use the “eq” keyword indicating “equals 445”.
access-list 140 deny   tcp any any eq 445 log
access-list 140 deny   udp any any eq 445 log
!
! We allow web traffic going to our web server address.  Here we use its address as destination, using
! the host shortcut instead of writing its equivalent:  192.168.3.2 0.0.0.0
! any source could produce a match.  We look then for any source, to the web server using ports used
! for HTTP (80) and HTTPS (443), since we use both.  These are both TCP protocols that
! use a three-level handshake, and originate in high ports (greater than 1024).
! We need these protocols since they will be used by GIAC's customers.
! It is important, though to think that we allowing the traffic, but the protocol might not be HTTP,
! or that traffic could convey attacks such as Code Red.  If CGI programs are not written well on
! GIAC's webserver, there could be problems, also.
! Also note that since we are doing NAT at this interface, we must specify the “real” private
! addresses instead of the “fake” public ones we will be announcing to the world.  Otherwise,
! the rules won't work.
access-list 140 permit tcp any host 192.168.3.2 eq 80 log
access-list 140 permit tcp any host 192.168.3.2 eq 443 log
!
! We allow mail protocols into the mail server.  Those protocols are SMTP (TCP port 25) and POP3
! (TCP port 110).  Again, we are opening ports that with defective server programs to answer them,
! could cause system compromise.  There have been serious bugs in SENDMAIL (a Unix SMTP server
! program) and QPOPPER (a Unix POP3 server program) that allowed remote superuser control
! over the server machine.
access-list 140 permit tcp any host 192.168.3.3 eq 25 log
access-list 140 permit tcp any host 192.168.3.3 eq 110 log
!
! Then, since we have a primary DNS for GIAC domain, we must allow DNS in (both TCP and UDP).
! DNS is a strange protocol.  It usually works in UDP, but if a request is large enough, it will spill over
! TCP packets.  There have been vulnerabilities in the BIND program (used in Unix to server DNS).
! Given that we have a primary DNS server, we'd better configure it so that it only gives the minimum
! information required.  We do not want a hacker to be given complete intelligence over our network!
access-list 140 permit tcp any host 192.168.4.2 eq 53 log
access-list 140 permit udp any host 192.168.4.2 eq 53 log
!
! We have to let the IPSec information pass though.  It was said that extended ACLs permitted
! selection of IP protocol.  ESP is one of them, so we will open things for it.
! Right now the rule specifies that any host can use ESP against our VPN box.  If there are few
! suppliers, we could just add their addresses here.
!
```

```
access-list 140 permit esp any host 120.0.0.2 log
! We will use IKE, for automatic key management.  It uses port 500 UDP.
access-list 140 permit udp any host 120.0.0.2 eq 500 log
!
! The previous rules were allowing several ports below 1024 belonging to some well-known protocols
! Since rules are checked from the first one, downward, we must provide specifics first, and
! general rules last.  Denying access to well-known ports is a common catch-all rule.  We don't
! like unauthorized access to any non-specified service that we might have forgotten to eliminate
! (not complying then with the defense-in-depth principle, but it is better to be safe than sorry).
! Please note that we are using "lt 1024", meaning any port less than 1024. We do it for both
! TCP and UDP.  In a couple of lines we are getting rid of NFS, finger, FTP and other services
! we might not have been interested to provide.
access-list 140 deny   tcp any any lt 1024 log
access-list 140 deny   udp any any lt 1024 log
!
! Then we allow answers to connections already established from the inside.  These are connections
! without the SYN bit set.  They are TCP by definition, since UDP doesn't keep state.  This is done
! this way, instead of specifying "permit ip any any" since it adds a layer of protection.  No TCP
! service would be permited on any port besides the ones we have already defined above.
access-list 140 permit tcp any any established
```

Then we have the outbound ACL. It will be assigned to the same interface, but with the "access-group 115 out" command, specifying outgoing packets. The ACL number is different than the one used for the incoming ACL. If we were just using cut and paste, and used the same number, it would be just appended to the other, and we would be left with only one access list.

```
! Outbound ACL
! Again, we will comment using the "!" character.
! First, we will block outgoing Microsoft networking protocols.  The same considerations as for
! incoming traffic apply here.
access-list 115 deny   tcp any any range 135 139
access-list 115 deny   udp any any range 135 netbios-ss
access-list 115 deny   udp any any eq 445
access-list 115 deny   tcp any any eq 445
! Let's get rid of outgoing IRC, since it is used by several intruders to report to base.
access-list 115 deny   tcp any any eq 6667 log
! Then we have to allow specific traffic.
! First we have DNS requests being made by our server.  As in the incoming ACL, allow both
! TCP and UDP DNS requests since both will be used.
access-list 115 permit tcp host 192.168.4.2 any eq 53
access-list 115 permit udp host 192.168.4.2 any eq 53
! DNS requests done to our server must go out also.
access-list 115 permit udp host 192.168.4.2 eq 53 any
access-list 115 permit tcp host 192.168.4.2 eq 53 any
! Outgoing traffic from our LAN.  We will fine-control it in the firewall.  So we will just
! indicate that any traffic masked through the firewall, going everywhere, will be permitted.
access-list 115 permit tcp host 192.168.1.2 any log
```

```
! Outgoing  traffic from the web server
access-list 115 permit tcp host 192.168.3.2 eq 80 any log
access-list 115 permit tcp host 192.168.3.2 eq 443 any log
! SMTP and POP traffic
access-list 115 permit tcp host 192.168.3.3 eq 25 any
access-list 115 permit tcp host 192.168.3.3 eq 110 any
! We must allow our mail server to function.  SMTP is used to send mail outside.
access-list 115 deny   permit host 192.168.3.3 any eq 25
! Just to be paranoid, and be a good network citizen, we block outgoing traffic from private
! addresses.  This in case something goes really wrong.
access-list 115 deny   ip any 10.0.0.0 0.255.255.255 log
access-list 115 deny   ip any 192.168.0.0 0.0.255.255 log
access-list 115 deny   ip any 172.16.0.0 0.15.255.255 log
!
! We have to let the IPSec information pass though.  It was said that extended ACLs permitted
! selection of IP protocol.  ESP is one of them, so we will open things for it.
! Right now the rule specifies that any host can use ESP against our VPN box.  If there are few
! suppliers, we could just add their addresses here.
!
access-list 115 permit esp host 120.0.0.2 any log
! We will use IKE, for automatic key management.  It uses port 500 UDP.
access-list 115 permit udp host 120.0.0.2 any eq 500 log
!
! Finally, what is not explicitly allowed is blocked.  The following line is the system default,
! but is better to write it down anyway, in order to make things clearer.
access-list 115 deny   ip any any log
```

Testing rules

One way to actually test some of these router ACLs, would be, for the mail connections (ports 25 and 110) in the inbound rules, to use nmap from a point outside our network and make a TCP scan toward the mailserver.

A machine running tcpdump would be placed just at the exit of the router interface that goes to the firewall, listening to all traffic.  It should only pick up traffic directed to ports 25 and 110 of the mailserver.

Besides, a *show ip access 140 command*, on the Cisco router, should report “hits” whenever we, from a point outside GIAC's network, access POP and SMTP services.  These hits should appear in both the incoming and outgoing ACL.  This command could be used for debugging purposes.

**External Firewall policy**

Our firewall is Linux/IPChains based.  To make configuration easier, a two-network IPChains setup tool was used, and its output modified by hand.  The tool was Tim Niemueller's IPchains Firewalling Webmin Module.

Whenever the system reboots, the firewall is set up using a script, with the whole configuration in it. It goes as follows:

```
#!/bin/sh
# IPchains Firewalling Script File
# Generated by IPchains Firewalling Webmin Module
# Copyright (C) 1999-2000 by Tim Niemueller, GPL
# http://www.niemueller.de/webmin/modules/ipchains/
# Created on 28/Aug/2001 17:50
# Modified for more interfaces by mserrano.

# Flush/clean everything
/sbin/ipchains -F
/sbin/ipchains -X

##MODE 1
##LEVEL HIGH
##MASQ
##FWTYPE ROUTER

# Set defaults to deny/reject everything not explicitly allowed.
/sbin/ipchains -P input DENY
/sbin/ipchains -P output DENY
/sbin/ipchains -P forward REJECT

/sbin/ipchains -A input -i lo -j ACCEPT
/sbin/ipchains -A output -i lo -j ACCEPT


#Do not accept packets from private class A on ext NIC
/sbin/ipchains -A input -i eth1 -s 10.0.0.0/8 -j DENY
/sbin/ipchains -A input -i eth1 -d 10.0.0.0/8 -j DENY
/sbin/ipchains -A output -i eth1 -s 10.0.0.0/8 -j DENY
/sbin/ipchains -A output -i eth1 -d 10.0.0.0/8 -j DENY

#Do not accept packets from private class B on ext NIC
/sbin/ipchains -A input -i eth1 -s 172.16.0.0/12 -j DENY
/sbin/ipchains -A input -i eth1 -d 172.16.0.0/12 -j DENY
/sbin/ipchains -A output -i eth1 -s 172.16.0.0/12 -j DENY
/sbin/ipchains -A output -i eth1 -d 172.16.0.0/12 -j DENY

#Do not accept packets from private class C on ext NIC
/sbin/ipchains -A input -i eth1 -s 192.168.0.0/16 -j DENY
/sbin/ipchains -A input -i eth1 -d 192.168.0.0/16 -j DENY
/sbin/ipchains -A output -i eth1 -s 192.168.0.0/16 -j DENY
/sbin/ipchains -A output -i eth1 -d 192.168.0.0/16 -j DENY
```

```
# Loopback packets should not be handled from ext NIC
/sbin/ipchains -A input -i eth1 -s 127.0.0.0/8 -j DENY
/sbin/ipchains -A output -i eth1 -s 127.0.0.0/8 -j DENY

#Refuse Bogus Broadcasts
/sbin/ipchains -A input -i eth1 -s 255.255.255.255 -j DENY
/sbin/ipchains -A input -i eth1 -d 0.0.0.0 -j DENY

# Refuse Requests from reserved IANA/ICANN adresses
/sbin/ipchains -A input -i eth1 -s 1.0.0.0/8 -j DENY
/sbin/ipchains -A input -i eth1 -s 2.0.0.0/8 -j DENY
/sbin/ipchains -A input -i eth1 -s 5.0.0.0/8 -j DENY
/sbin/ipchains -A input -i eth1 -s 7.0.0.0/8 -j DENY
/sbin/ipchains -A input -i eth1 -s 23.0.0.0/8 -j DENY
/sbin/ipchains -A input -i eth1 -s 27.0.0.0/8 -j DENY
/sbin/ipchains -A input -i eth1 -s 31.0.0.0/8 -j DENY
/sbin/ipchains -A input -i eth1 -s 36.0.0.0/8 -j DENY
/sbin/ipchains -A input -i eth1 -s 37.0.0.0/8 -j DENY
/sbin/ipchains -A input -i eth1 -s 39.0.0.0/8 -j DENY
/sbin/ipchains -A input -i eth1 -s 41.0.0.0/8 -j DENY
/sbin/ipchains -A input -i eth1 -s 42.0.0.0/8 -j DENY
/sbin/ipchains -A input -i eth1 -s 58.0.0.0/8 -j DENY
/sbin/ipchains -A input -i eth1 -s 59.0.0.0/8 -j DENY
/sbin/ipchains -A input -i eth1 -s 60.0.0.0/8 -j DENY
/sbin/ipchains -A input -i eth1 -s 67.0.0.0/8 -j DENY
/sbin/ipchains -A input -i eth1 -s 218.0.0.0/8 -j DENY
/sbin/ipchains -A input -i eth1 -s 219.0.0.0/8 -j DENY
/sbin/ipchains -A input -i eth1 -s 68.0.0.0/6 -j DENY
/sbin/ipchains -A input -i eth1 -s 72.0.0.0/5 -j DENY
/sbin/ipchains -A input -i eth1 -s 80.0.0.0/4 -j DENY
/sbin/ipchains -A input -i eth1 -s 96.0.0.0/3 -j DENY
/sbin/ipchains -A input -i eth1 -s 220.0.0.0/6 -j DENY

# Basic ICMP packages are needed for running a network
/sbin/ipchains -A input -i eth1 -p icmp --icmp-type source-quench -d 120.0.0.130 -j ACCEPT
/sbin/ipchains -A output -i eth1 -p icmp --icmp-type source-quench -d 0.0.0.0/0 -j ACCEPT
/sbin/ipchains -A input -i eth1 -p icmp --icmp-type parameter-problem -d 120.0.0.130 -j ACCEPT
/sbin/ipchains -A output -i eth1 -p icmp --icmp-type parameter-problem -d 0.0.0.0/0 -j ACCEPT
/sbin/ipchains -A input -i eth1 -p icmp --icmp-type destination-unreachable -d 120.0.0.130 -j ACCEPT
/sbin/ipchains -A output -i eth1 -p icmp --icmp-type destination-unreachable -d 0.0.0.0/0 -j DENY
/sbin/ipchains -A input -i eth1 -p icmp --icmp-type time-exceeded -d 120.0.0.130 -j ACCEPT
/sbin/ipchains -A output -i eth1 -p icmp --icmp-type time-exceeded -d 0.0.0.0/0 -j ACCEPT


##=> DNS-inout
##-> This allows a host in the internal network to lookup hostnames by
```

##-> querying external nameservers.  Used for internal DNS servers.  Uses masquerading.

/sbin/ipchains -A input -i eth0 -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 53 -p udp -j ACCEPT
/sbin/ipchains -A output -i eth0 -s ! 192.168.6.1 53 -d 192.168.6.0/255.255.255.0 1024:65535 -p udp -j ACCEPT
/sbin/ipchains -A input -i eth1 -s ! 192.168.6.1 53 -d 192.168.6.0/255.255.255.0 1024:65535 -p udp -j ACCEPT
/sbin/ipchains -A input -i eth1 -s ! 192.168.6.1 53 -d 120.0.0.130 1024:65535 -p udp -j ACCEPT
/sbin/ipchains -A output -i eth1 -s 120.0.0.130 1024:65535 -d ! 192.168.6.1 53 -p udp -j ACCEPT
/sbin/ipchains -A forward -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 53 -p udp -j MASQ

##=> DNS-outfw
##-> Allows clients on the outside network to access the primary DNS server.
##->
/sbin/ipchains -A input -i eth1 -s ! 192.168.6.1 1024:65535 -d 192.168.4.2 53 -p udp -j ACCEPT
/sbin/ipchains -A output -i eth1 -s 192.168.4.2 53 -d ! 192.168.6.1 1024:65535 -p udp -j ACCEPT
/sbin/ipchains -A output -i eth2 -s ! 192.168.6.1 1024:65535 -d 192.168.4.2 53 -p udp -j ACCEPT
/sbin/ipchains -A input -i eth2 -s 192.168.4.2 53 -d ! 192.168.6.1 1024:65535 -p udp -j ACCEPT
/sbin/ipchains -A forward -i eth1 -s 192.168.4.2 53 -d ! 192.168.6.1 1024:65535 -p udp -j ACCEPT
/sbin/ipchains -A forward -i eth2 -s ! 192.168.6.1 1024:65535 -d 192.168.4.2 53 -p udp -j ACCEPT

##=> DNS-Mail
##-> Allows the mail server to access the primary DNS server.
##->
/sbin/ipchains -A input -i eth4 -s 192.168.3.3 1024:65535 -d 192.168.4.2 53 -p udp -j ACCEPT
/sbin/ipchains -A output -i eth4 -s 192.168.4.2 53 -d 192.168.3.3 1024:65535 -p udp -j ACCEPT
/sbin/ipchains -A output -i eth2 -s 192.168.3.3 1024:65535 -d 192.168.4.2 53 -p udp -j ACCEPT
/sbin/ipchains -A input -i eth2 -s 192.168.4.2 53 -d 192.168.3.3 1024:65535 -p udp -j ACCEPT
/sbin/ipchains -A forward -i eth4 -s 192.168.4.2 53 -d 192.168.3.3 1024:65535 -p udp -j ACCEPT
/sbin/ipchains -A forward -i eth2 -s 192.168.3.3 1024:65535 -d 192.168.4.2 53 -p udp -j ACCEPT

##=> DNS-fwin
##-> Allows the firewall host to use a DNS server on the inside network to
##-> resolve names and adresses.
/sbin/ipchains -A output -i eth0 -s 192.168.6.1 1024:65535 -d 192.168.6.0/255.255.255.0 53 -p udp -j ACCEPT
/sbin/ipchains -A input -i eth0 -s 192.168.6.0/255.255.255.0 53 -d 192.168.6.1 1024:65535 -p udp -j ACCEPT

##=> FTP.Active-inout
##-> Allows clients on the internal network to access external FTP servers
##-> via active FTP. This is more secure than passive FTP but is still a risk

```
##-> because FTP passwords are transferred as clear text!
/sbin/ipchains -A input -i eth0 -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 21 -p tcp -j
ACCEPT
/sbin/ipchains -A output -i eth0 -s ! 192.168.6.1 21 -d 192.168.6.0/255.255.255.0 1024:65535 ! -y -p tcp -j
ACCEPT
/sbin/ipchains -A input -i eth0 -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 20 ! -y -p tcp -j
ACCEPT
/sbin/ipchains -A output -i eth0 -s ! 192.168.6.1 20 -d 192.168.6.0/255.255.255.0 1024:65535 -p tcp -j
ACCEPT
if [ -e /lib/modules/$(uname -r)/ipv4/ip_masq_ftp.o ]; then
  if [ -x /sbin/insmod ]; then
    if ! $(grep -s ip_masq_ftp /proc/modules >/dev/null); then
      /sbin/insmod -p -s ip_masq_ftp
    fi
  fi
fi
/sbin/ipchains -A input -i eth1 -s ! 192.168.6.1 21 -d 120.0.0.130 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth1 -s 120.0.0.130 1024:65535 -d ! 192.168.6.1 21 -p tcp -j ACCEPT
/sbin/ipchains -A forward -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 21 -p tcp -j MASQ
/sbin/ipchains -A input -i eth1 -s ! 192.168.6.1 20 -d 120.0.0.130 1024:65535 -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth1 -s 120.0.0.130 1024:65535 -d ! 192.168.6.1 20 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A forward -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 20 -p tcp -j MASQ


##=> HTTP-inout
##-> Allows clients on the internal network to surf the web, and access our www server too!
##     Also allows clients on the external network to see our www server.

/sbin/ipchains -A input -i eth0 -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 80 -p tcp -j
ACCEPT
/sbin/ipchains -A output -i eth0 -s ! 192.168.6.1 80 -d 192.168.6.0/255.255.255.0 1024:65535 ! -y -p tcp -j
ACCEPT
/sbin/ipchains -A input -i eth1 -s ! 192.168.6.1 80 -d 120.0.0.130 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth1 -s 120.0.0.130 1024:65535 -d ! 192.168.6.1 80 -p tcp -j ACCEPT
/sbin/ipchains -A input -i eth1 -s 0.0.0.0/0.0.0.0 1024:65535 -d 192.168.3.2 80 -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth1 -s 192.168.3.2 80 -d 0.0.0.0/0.0.0.0 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A input -i eth4 -s 192.168.3.2 80 -d ! 192.168.6.1 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth4 -s ! 192.168.6.1 1024:65535 -d 192.168.3.2 80 -p tcp -j ACCEPT
# Order is important here, since masquerading is the last option for all internal accesses to the web
# except for the ones going to our webserver
/sbin/ipchains -A forward -i eth4 -s 0.0.0.0/0.0.0.0 1024:65535 -d 192.168.3.2 80 -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth0 -s 192.168.3.2 80 -d ! 192.168.6.1 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth1 -s 192.168.3.2 80 -d ! 192.168.6.1 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A forward -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 80 -p tcp -j MASQ


##=> HTTPS-inout
```

##-> Allows clients on the internal network to surf the web, and access our www server too!
##  Also allows clients on the external network to see our www server.  Both using HTTPS.

/sbin/ipchains -A input -i eth0 -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 443 -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth0 -s ! 192.168.6.1 443 -d 192.168.6.0/255.255.255.0 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A input -i eth1 -s ! 192.168.6.1 443 -d 120.0.0.130 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth1 -s 120.0.0.130 1024:65535 -d ! 192.168.6.1 443 -p tcp -j ACCEPT
/sbin/ipchains -A input -i eth1 -s 0.0.0.0/0.0.0.0 1024:65535 -d 192.168.3.2 443 -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth1 -s 192.168.3.2 443 -d 0.0.0.0/0.0.0.0 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A input -i eth4 -s 192.168.3.2 443 -d ! 192.168.6.1 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth4 -s ! 192.168.6.1 1024:65535 -d 192.168.3.2 443 -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth4 -s 0.0.0.0/0.0.0.0 1024:65535 -d 192.168.3.2 443 -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth0 -s 192.168.3.2 443 -d ! 192.168.6.1 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth1 -s 192.168.3.2 443 -d ! 192.168.6.1 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A forward -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 443 -p tcp -j MASQ

##=> Traffic from the VPN.
## We just remember that traffic comes from the 192.168.7.0/24 network on the other side.

/sbin/ipchains -A input -i eth3 -s 192.168.7.0/24 1024:65535 -d 192.168.3.2 80 -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth3 -s 192.168.3.2 80 -d 192.168.7.0/24 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A input -i eth4 -s 192.168.3.2 80 -d 192.168.7.0/24 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth4 -s 192.168.7.0/24 1024:65535 -d 192.168.3.2 80 -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth4 -s 192.168.7.0/24 1024:65535 -d 192.168.3.2 80 -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth3 -s 192.168.3.2 80 -d 192.168.7.0/24 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A input -i eth3 -s 192.168.7.0/24 1024:65535 -d 192.168.3.2 443 -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth3 -s 192.168.3.2 443 -d 192.168.7.0/24 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A input -i eth4 -s 192.168.3.2 443 -d 192.168.7.0/24 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth4 -s 192.168.7.0/24 1024:65535 -d 192.168.3.2 443 -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth4 -s 192.168.7.0/24 1024:65535 -d 192.168.3.2 443 -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth3 -s 192.168.3.2 443 -d 192.168.7.0/24 1024:65535 ! -y -p tcp -j ACCEPT

##=> POP3-inout
##-> Allows clients on the internal network to fetch emails from our POP3 server.
##-> Also provides for external client access to the POP3 server.

/sbin/ipchains -A input -i eth0 -s 192.168.6.0/255.255.255.0 1024:65535 -d 192.168.3.3 110 -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth0 -s 192.168.3.3 110 -d 192.168.6.0/255.255.255.0 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A input -i eth1 -s 0.0.0.0/0.0.0.0 1024:65535 -d 192.168.3.3 110 -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth1 -s 192.168.3.3 110 -d 0.0.0.0/0.0.0.0 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A input -i eth4 -s ! 192.168.3.3 110 -d 0.0.0.0/0.0.0.0 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth4 -s 0.0.0.0/0.0.0.0 1024:65535 -d 192.168.3.3 110 -p tcp -j ACCEPT

/sbin/ipchains -A forward -i eth4 -s 0.0.0.0/0.0.0.0 1024:65535 -d 192.168.3.3 110 -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth1 -s 192.168.3.3 110 -d 0.0.0.0/0.0.0.0 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth0 -s 192.168.3.3 110 -d 192.168.6.0/255.255.255.0 1024:65535 ! -y -p tcp -j ACCEPT

##=> Ping-infw
##-> Allows host on the inside network to ping the firewall host. Usually you
##-> want that to check the connection between computers and the firewall or
##-> if the firewall host is up.
/sbin/ipchains -A input -i eth0 -s 192.168.6.0/255.255.255.0 -d 192.168.6.1 -p icmp --icmp-type echo-request -j ACCEPT
/sbin/ipchains -A output -i eth0 -s 192.168.6.1 -d 192.168.6.0/255.255.255.0 -p icmp --icmp-type echo-reply -j ACCEPT

##=> Ping-inout
##-> Allows clients on the internal network to ping other machines on the
##-> external network.
/sbin/ipchains -A input -i eth0 -s 192.168.6.0/255.255.255.0 -d ! 192.168.6.1 -p icmp --icmp-type echo-request -j ACCEPT
/sbin/ipchains -A output -i eth0 -s ! 192.168.6.1 -d 192.168.6.0/255.255.255.0 -p icmp --icmp-type echo-reply -j ACCEPT
/sbin/ipchains -A input -i eth1 -s ! 192.168.6.1 -d 120.0.0.130 -p icmp --icmp-type echo-reply -j ACCEPT
/sbin/ipchains -A output -i eth1 -s 120.0.0.130 -d ! 192.168.6.1 -p icmp --icmp-type echo-request -j ACCEPT
/sbin/ipchains -A forward -s 192.168.6.0/255.255.255.0 -d ! 192.168.6.1 -p icmp --icmp-type echo-request -j MASQ

##=> Ping-fwin
##-> Allows the firewall to ping hosts on the inside network.
/sbin/ipchains -A output -i eth0 -s 192.168.6.1 -d 192.168.6.0/255.255.255.0 -p icmp --icmp-type echo-request -j ACCEPT
/sbin/ipchains -A input -i eth0 -s 192.168.6.0/255.255.255.0 -d 192.168.6.1 -p icmp --icmp-type echo-reply -j ACCEPT

##=> SMTP-inout
##-> Allows clients on the internal network to send emails through our SMTP
##-> server.  Also provides for external contact to our SMTP server.
/sbin/ipchains -A input -i eth0 -s 192.168.6.0/255.255.255.0 1024:65535 -d 192.168.3.3 25 -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth0 -s 192.168.3.3 25 -d 192.168.6.0/255.255.255.0 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A input -i eth1 -s 0.0.0.0/0.0.0.0 1024:65535 -d 192.168.3.3 25 -p tcp -j ACCEPT

/sbin/ipchains -A output -i eth1 -s 192.168.3.3 25 -d 0.0.0.0/0.0.0.0 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A input -i eth4 -s ! 192.168.3.3 25 -d 0.0.0.0/0.0.0.0 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth4 -s 0.0.0.0/0.0.0.0 1024:65535 -d 192.168.3.3 25 -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth4 -s 0.0.0.0/0.0.0.0 1024:65535 -d 192.168.3.3 25 -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth1 -s 192.168.3.3 25 -d 0.0.0.0/0.0.0.0 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth0 -s 192.168.3.3 25 -d 192.168.6.0/255.255.255.0 1024:65535 ! -y -p tcp -j ACCEPT

##=> SMTP-mailserver
##-> Allows our mailserver to access other SMTP servers
##-> on the external network.
/sbin/ipchains -A input -i eth4 -s 192.168.3.3 1024:65535 -d ! 192.168.6.1 25 -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth4 -s ! 192.168.6.1 25 -d 192.168.3.3 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A input -i eth1 -s ! 192.168.6.1 25 -d 192.168.3.3 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth1 -s 192.168.3.3 1024:65535 -d ! 192.168.6.1 25 -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth1 -s 192.168.3.3 1024:65535 -d ! 192.168.6.1 25 -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth4 -s ! 192.168.6.1 25 -d 192.168.3.3 1024:65535 ! -y -p tcp -j ACCEPT

##=> WWW to database
##-> Allows our webserver to access the PostgreSQL database
/sbin/ipchains -A input -i eth4 -s 192.168.3.3 1024:65535 -d 192.168.2.2 5423 -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth4 -s 192.168.2.2 5423 -d 192.168.3.3 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A input -i eth5 -s 192.168.2.2 5423 -d 192.168.3.3 1024:65535 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth5 -s 192.168.3.3 1024:65535 -d 192.168.2 5423 -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth5 -s 192.168.3.3 1024:65535 -d 192.168.6.1 5423 -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth4 -s 192.168.2.2 5423 -d 192.168.3.3 1024:65535 ! -y -p tcp -j ACCEPT

##=> SSH-infw
##-> Allows SSH conntections from the inside network to the firewall.
/sbin/ipchains -A input -i eth0 -p tcp -s 192.168.6.0/255.255.255.0 1024:65535 -d 192.168.6.1 22 -j ACCEPT
/sbin/ipchains -A output -i eth0 -p tcp ! -y -s 192.168.6.1 22 -d 192.168.6.0/255.255.255.0 1024:65535 -j ACCEPT
/sbin/ipchains -A input -i eth0 -p tcp -s 192.168.6.0/255.255.255.0 513:1023 -d 192.168.6.1 22 -j ACCEPT
/sbin/ipchains -A output -i eth0 -p tcp ! -y -s 192.168.6.1 22 -d 192.168.6.0/255.255.255.0 513:1023 -j ACCEPT

##=> SSH-inout
##-> Allows clients on the internal network to connect to secure shell
##-> servers on any network.  There are lots of rules since there are lots of interfaces!.

/sbin/ipchains -A input -i eth0 -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 22 -p tcp -j ACCEPT

```
/sbin/ipchains -A output -i eth0 -s ! 192.168.6.1 22 -d 192.168.6.0/255.255.255.0 1024:65535 ! -y -p tcp -j
ACCEPT
/sbin/ipchains -A input -i eth0 -s 192.168.6.0/255.255.255.0 513:1023 -d ! 192.168.6.1 22 -p tcp -j
ACCEPT
/sbin/ipchains -A output -i eth0 -s ! 192.168.6.1 22 -d 192.168.6.0/255.255.255.0 513:1023 ! -y -p tcp -j
ACCEPT
/sbin/ipchains -A input -i eth1 -s ! 192.168.6.1 22 -d 192.168.6.0/255.255.255.0 1024:65535 ! -y -p tcp -j
ACCEPT
/sbin/ipchains -A output -i eth1 -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 22 -p tcp -j
ACCEPT
/sbin/ipchains -A forward -i eth1 -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 22 -p tcp -j
ACCEPT
/sbin/ipchains -A input -i eth2 -s ! 192.168.6.1 22 -d 192.168.6.0/255.255.255.0 1024:65535 ! -y -p tcp -j
ACCEPT
/sbin/ipchains -A output -i eth2 -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 22 -p tcp -j
ACCEPT
/sbin/ipchains -A forward -i eth2 -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 22 -p tcp -j
ACCEPT
/sbin/ipchains -A input -i eth3 -s ! 192.168.6.1 22 -d 192.168.6.0/255.255.255.0 1024:65535 ! -y -p tcp -j
ACCEPT
/sbin/ipchains -A output -i eth3 -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 22 -p tcp -j
ACCEPT
/sbin/ipchains -A forward -i eth3 -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 22 -p tcp -j
ACCEPT
/sbin/ipchains -A input -i eth4 -s ! 192.168.6.1 22 -d 192.168.6.0/255.255.255.0 1024:65535 ! -y -p tcp -j
ACCEPT
/sbin/ipchains -A output -i eth4 -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 22 -p tcp -j
ACCEPT
/sbin/ipchains -A forward -i eth4 -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 22 -p tcp -j
ACCEPT
/sbin/ipchains -A input -i eth5 -s ! 192.168.6.1 22 -d 192.168.6.0/255.255.255.0 1024:65535 ! -y -p tcp -j
ACCEPT
/sbin/ipchains -A output -i eth5 -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 22 -p tcp -j
ACCEPT
/sbin/ipchains -A forward -i eth5 -s 192.168.6.0/255.255.255.0 1024:65535 -d ! 192.168.6.1 22 -p tcp -j
ACCEPT
/sbin/ipchains -A forward -i eth0 -s ! 192.168.6.1 22 -d 192.168.6.0/255.255.255.0 1024:65535 ! -y -p tcp -
j ACCEPT
/sbin/ipchains -A input -i eth1 -s ! 192.168.6.1 22 -d 192.168.6.0/255.255.255.0 513:1023 ! -y -p tcp -j
ACCEPT
/sbin/ipchains -A output -i eth1 -s 192.168.6.0/255.255.255.0 513:1023 -d ! 192.168.6.1 22 -p tcp -j
ACCEPT
/sbin/ipchains -A forward -i eth1 -s 192.168.6.0/255.255.255.0 513:1023 -d ! 192.168.6.1 22 -p tcp -j
ACCEPT
/sbin/ipchains -A input -i eth2 -s ! 192.168.6.1 22 -d 192.168.6.0/255.255.255.0 513:1023 ! -y -p tcp -j
ACCEPT
/sbin/ipchains -A output -i eth2 -s 192.168.6.0/255.255.255.0 513:1023 -d ! 192.168.6.1 22 -p tcp -j
```

ACCEPT
/sbin/ipchains -A forward -i eth2 -s 192.168.6.0/255.255.255.0 513:1023 -d ! 192.168.6.1 22 -p tcp -j ACCEPT
/sbin/ipchains -A input -i eth3 -s ! 192.168.6.1 22 -d 192.168.6.0/255.255.255.0 513:1023 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth3 -s 192.168.6.0/255.255.255.0 513:1023 -d ! 192.168.6.1 22 -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth3 -s 192.168.6.0/255.255.255.0 513:1023 -d ! 192.168.6.1 22 -p tcp -j ACCEPT
/sbin/ipchains -A input -i eth4 -s ! 192.168.6.1 22 -d 192.168.6.0/255.255.255.0 513:1023 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth4 -s 192.168.6.0/255.255.255.0 513:1023 -d ! 192.168.6.1 22 -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth4 -s 192.168.6.0/255.255.255.0 513:1023 -d ! 192.168.6.1 22 -p tcp -j ACCEPT
/sbin/ipchains -A input -i eth5 -s ! 192.168.6.1 22 -d 192.168.6.0/255.255.255.0 513:1023 ! -y -p tcp -j ACCEPT
/sbin/ipchains -A output -i eth5 -s 192.168.6.0/255.255.255.0 513:1023 -d ! 192.168.6.1 22 -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth5 -s 192.168.6.0/255.255.255.0 513:1023 -d ! 192.168.6.1 22 -p tcp -j ACCEPT
/sbin/ipchains -A forward -i eth0 -s ! 192.168.6.1 22 -d 192.168.6.0/255.255.255.0 513:1023 ! -y -p tcp -j ACCEPT

There are other things in the Linux firewall configuration that have to be considered:

- Source routing blocking.
- SYN cookies (a form of protection against SYN floods).

**VPN Policy**

Smoothwall, the solution used, supports Linux Free/SWAN 1.9. The configuration is done using a simple web interface. The software comes preconfigured for the following:

- Automatic key exchange using IKE.
- Encryption algorithm used: Triple DES, only.
- Use of shared secrets to authenticate machines during key negotiations.

ESP will be used, since it is the one that actually encrypts packets. We will not be satisfied with authentication, only.

We will assume that the supplier's VPN gateway address is 123.0.0.2, its router is 123.0.0.1 and the private supplier's network is 192.168.7.0/24.

Our **ipsec.secrets** file contains:

120.0.0.2 123.0.0.2 : PSK "long_secret_random_phrase_with_lots_of_characters”

Our **ipsec.conf** file has:

```
config setup
        interfaces=%defaultroute
        klipsdebug=none
        plutodebug=none
        plutoload=%search
        plutostart=%search
        uniqueids=yes
# Retry forever
      keyingtries=0
conn supplier
 # left security gateway (public-network address)
        left=120.0.0.2
        # next hop to reach right
        leftnexthop=120.0.0.1
        # subnet behind left (omit if there is no subnet)
        leftsubnet=192.168.5.0/24
        # right s.g., subnet behind it, and next hop to reach left
        right=123.0.0.2
        rightnexthop=123.0.0.1
        rightsubnet=192.168.7.0/24
        auto=start
```

Since there is no more data, defaults rule. Free/SWAN uses IKE for key exchange. Authenticates using ESP, has 8 hours of key life and the IKE lifetime is one hour.

III - Audit Your Security Architecture

PlanningGIAC's firewall has six interfaces. An audit of it should be considering these in a connectivity matrix, where from each interface, connections should be tried toward each of the others. There are 30 different combinations possible. Successful connections should be registered and checked against the desired behavior.

The automated way to perform the audit would be using a port scanner. Nmap (http://www.insecure.org/nmap/) uses several approaches for this. It is possible to scan for TCP ports in several ways, using a variety of header flags. UDP can be checked also. With only a single command, a whole network can be checked for open TCP or UDP ports.

How much would it cost? Nmap runs on top of Linux. What would be required, in terms of hardware, is a mobile computer (laptop) costing $1500, running Linux. Other costs are labor-related. The test is fairly automated. The biggest effort required, besides planning, is related to the movement of the test laptop among interfaces (six changes), the processing of information, and check of service continuation after the tests.

When to plan for such an audit? Unfortunately, port scanning can be disruptive. In the past, scanning has been known to freeze routers and other devices. Port scanning must be planned, therefore, in such a way as to minimize its impact on heavy usage. If GIAC Enterprises' traffic is lower during the weekend, for example, tests should start early Saturday morning. Shall the test take too long, it could be paused and resumed the next weekend.

Another reason to avoid system usage is related to the laptop placement. If there are no switch ports available for its usage on a given network, a small hub or switch should be placed, causing a slight disruption in traffic flow.

Right from the start, though, due to the nature of the solution being used, we know that it is possible to fool the firewall using IP packet fragmentation. IPChains uses simple packet filtering. If a second packet arrives, claiming to be the next fragment of an already accepted one, it will be accepted automatically, regardless of offsets overwriting, maybe, the old packet's header itself.

**Audit results**

As mentioned above, the tool that will be used is nmap.

Checks are be done using both TCP and UDP portscanning. To check UDP ports in the webserver, the command is

nmap -sU 192.168.3.2/32

In order to check TCP ports, nmap offers several options: normal TCP connect scan (-sT), SYN scan (-sS), FIN (-sF), Xmas (-sX), null scans (-sN). If the packet filtering rules are simple enough, there would be differences, due to the ways option flags in TCP packets interact with them. Trial runs were done then, with these options.

The report is that all these different TCP-scanning options produce exactly the same results. For example, with the webserver, only ports 80 and 443 appear open in a quick portscan:

```
Starting nmap V. 2.54BETA7 ( www.insecure.org/nmap/ )
Interesting ports on  (192.168.3.2):
(The 419 ports scanned but not shown below are in state: filtered)
Port       State       Service
80/tcp     open        http
443/tcp    open        https

Nmap run completed -- 1 IP address (1 host up) scanned in 415 seconds
```

What this means is that the elaborate scheme of INPUT, OUTPUT and FORWARD chains, configured with SYN-bit discrimination successfully eliminate the possibility of intelligence gathering by complex portscanning. The downside is that rules are unmanageable. More than 190 rules make error detection, and network modifications almost heroic in nature.

Results:
192.168.3.2 is reachable for both ports 80 and 443 TCP from the external, internal and VPN interfaces.
192.168.3.3 is reachable from the internal and external interfaces for both ports 25 and 110 TCP
192.168.4.2 is reachable from the external interface.

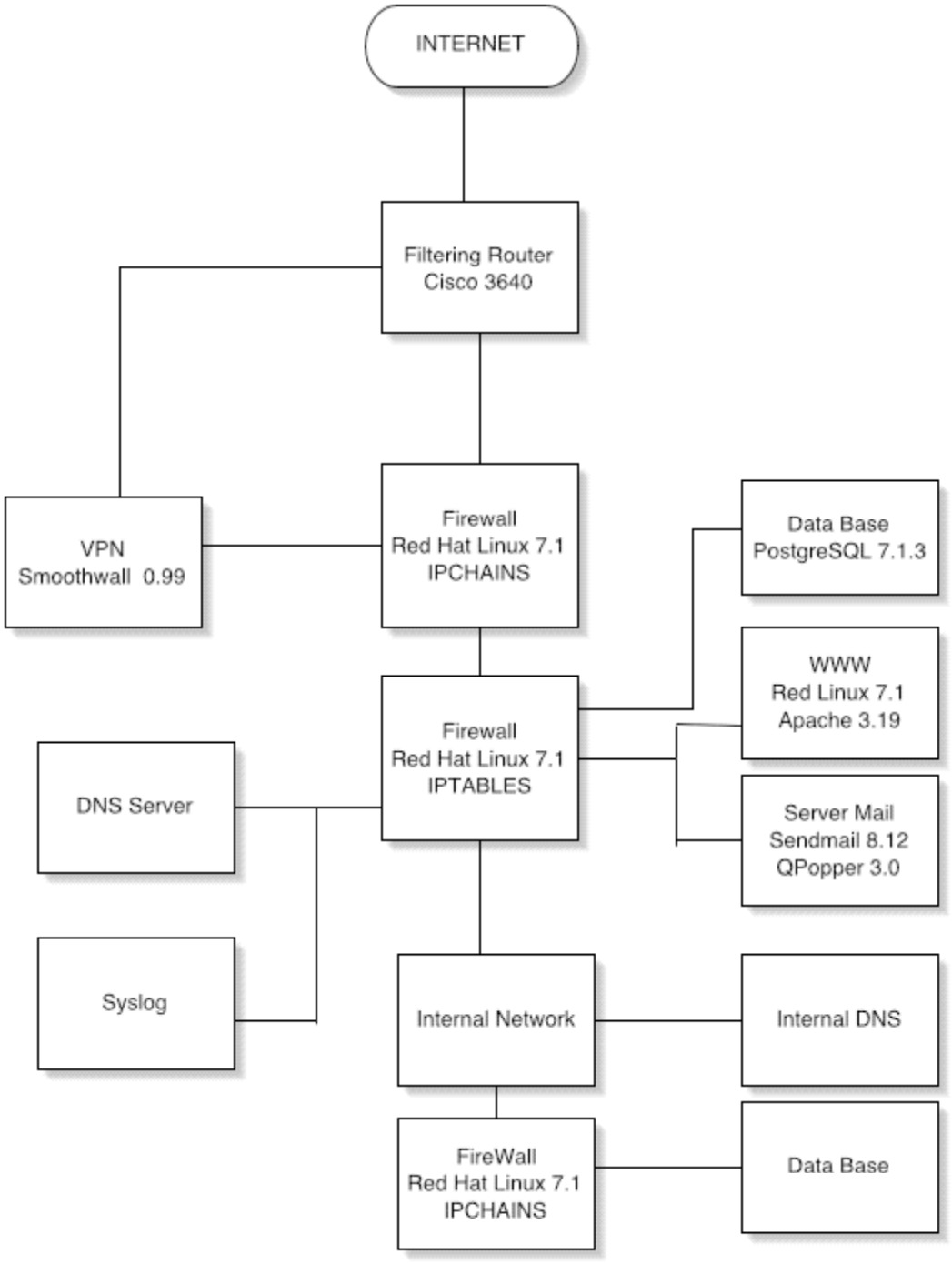
In normal operations, services are reachable.

**Audit evaluation**

As it has been already mentioned, the firewall blocks elaborate TCP scans. These yield no extra information.

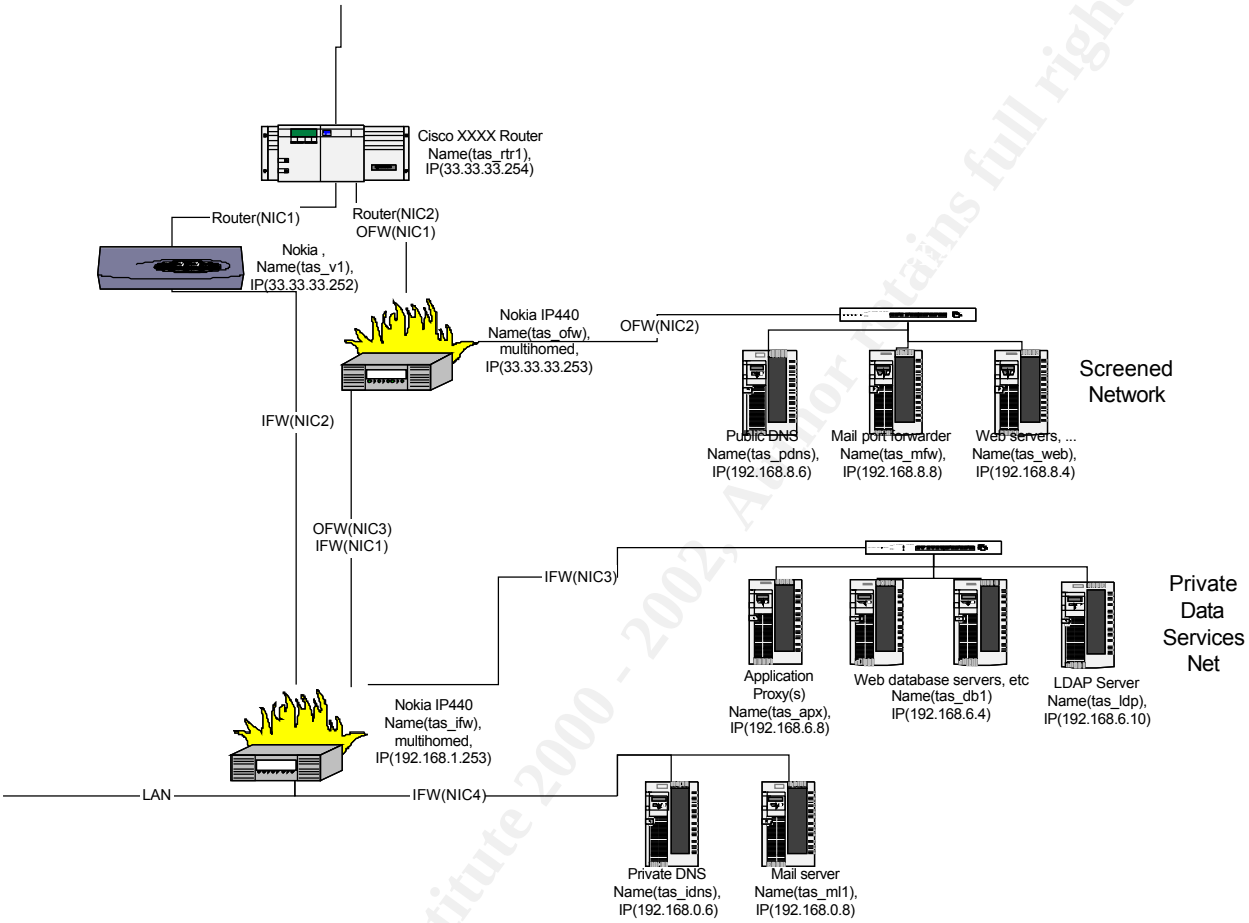
One problem with the audit methodology used, is that the only way to produce complete results is to substitute even the servers with the nmap station. There are ports that are open exclusively for certain addresses, and that could be checked only by using them to perform the scans. A test such as the one described could miss extra open ports from the servers' point of view. A more thorough test then would need IP impersonation, and TCPDUMP running on a machine that could intercept TCP handshake replies. Tests could be run only at very light usage times.

What is clear is the complexity of the rules. A rules check takes a long time. One way to reduce complexity, and to increase security levels would add an extra firewall to the design. A stateful IPTables firewall would inherit all interfaces except the VPN one from the current one. The IPChains firewall would then be doing quick filtering and would remain with the illegal address removal. This option is shown in the following graphic.



IV - Design Under Fire.

For this part, we will consider Scott Marshall's design. The paper's location is http://www.sans.org/y2k/practical/Scott_Marshall_GCFW.zip




It uses Nokia IP440 firewalls, which he states, run CheckPoint's Firewall-1 4.1 SP2 on top of BSDI. For the sake of this exercise, we assume that the system was installed and forgotten with that old version of the firewall software.

Firewall vulnerabilities

A quick look at SecurityFocus.com reveals several vulnerabilities for that particular version:

- 2001-09-12: Check Point Firewall-1 GUI Log Viewer Vulnerability
- 2001-09-08: Check Point Firewall-1 GUI Client Log Viewer Symbolic Link Vulnerability
- 2001-07-18: Check Point Firewall-1 SecureRemote Network Information Leak Vulnerability
- 2001-07-11: Check Point Firewall-1/VPN-1 Management Station Format String Vulnerability
- 2001-01-17: Check Point Firewall-1 4.1 Denial of Service Vulnerability
- 2000-12-14: Check Point Firewall-1 Fast Mode TCP Fragment Vulnerability

A description of three of these follows:

Regarding, BugTraq bug id 2238, Check Point Firewall-1 4.1 Denial of Service Vulnerability, Security Focus states the following:

> A problem with the license manager used with the Firewall-1 package could allow a Denial of Service. The problem manifests itself when the internal interface receives a large number of packets that are source routed and containing ficticious (or even valid) addresses. In a system containing a license with a limited number of protected IP addresses, the license manager calculates the address space protected by counting the number of addresses crossing the internal interface. When the large number of packets cross the internal interface, each IP address is added to the number calculated under license coverage. When the number of covered IP addresses is exceeded, an error message is generated on the console for each IP address outside of the covered range. With each error message generated, the load on the Firewall system CPU raises. This makes it possible for a user with malicious motives to make a firewall system inaccessible from the console by sending a large number of IP addresses to the internal interface. (SecurityFocus, 2238)

This is clearly a type of vulnerability that cannot be exploited from the outside. Nevertheless, its existance could cause problems with dangerous users.

There is another vulnerability. BugTraq id 3336, Check Point Firewall-1 GUI Log Viewer Vulnerability, is described by SecurityFocus, as follows:

> It has been reported that Firewall-1 may contain a buffer overflow vulnerability. The vulnerability is allegedly in logging of authentication attempts by GUI log viewing clients.
> It may be possible for remote attackers to execute arbitrary code as root on systems running Firewall-1.
> The attack must be launched from hosts who are permitted to view logs via the GUI interface. (SecurityFocus, 3336)

This one is almost useless for attack purposes. Substantial permissions are required to exploit it.

A third vulnerability, that receives BugTraq's id 2143 is the Fast Mode TCP Fragment Vulnerability. SecurityFocus describes it:

> Check Point Software's VPN-1 and Firewall-1 products contain a vulnerability in their "Fast Mode" option that may allow an attacker to bypass access control restrictions and access certain blocked services. Fast Mode is a setting that turns off analysis of packets in tcp sessions after the TCP 3-way handshake has completed for speed-crtitical services.
> If this setting is enabled on a firewall, it may be possible for a remote attacker to access blocked services on the host protected by the firewall using fastmode. It is also reportedly possible to access hosts at least one hop away on the same interface as the target host being protected.
> In order for this to be possible, at least one TCP service on a host protected by the firewall must be accessible by the attacker to which a SYN can be sent legitimately. The vulnerability is due to a failure to handle malformed fragmented TCP segments.

> This vulnerability may allow attackers to access vulnerable services normally protected by the firewall ruleset. (SecurityFocus, 2143)

Among these, one that can be easily used for a theoretical attack against the firewall (at least the denial of service flavor) is the first one. The last one could be used only if the firewall is configured to use Fast Mode (by default CheckPoint isn't).

Any internal user could generate those source routed packets and hit the licence limit, increase the CPU load and force reboot of the firewall due to complete lack of access to the console. Of course, that implies:

- The attack is internal. Only one machine belonging to the enterprise can be used for it.
- The firewall is not of the unlimited flavor. CheckPoint has several license sizes: several of them limited.
- The user installs a program that can build IP packets, and source-route them to the firewall.

If the administrator is unable to upgrade the system to prevent this Denial-Of-Service attack, one quick fix could be to install a Linux machine with two interfaces just in front of the interface, and eliminate source-routed packets there.

**Denial of service attack using ICMP floods**

The design has only one connection to the Internet, therefore flooding from several compromised machines can saturate that link and put the network out of business.

Countermeasures? Well, there can be several, just to mitigate effects, but no complete cure is available. For starts, the ISP must take part: it could filter reserved ICANN addresses, so as to mitigate spoofing. Also Cisco's CAR (Commited Access Rate) technology could be used to limit the rate incoming ICMP packets arrive at. In case ICMP floods do not use spoofing, the ISP could prevent that traffic from reaching the link, using access control lists in its routers. It could become an arms race, in which the ISP technicians would have to add new filters continuously.

One way to make the network more resilient would require extra bandwitdth (but it would be a short matter of time for the script kiddie to grab more machines and use them against the network).

Another network link with another ISP would help. The other ISP would also be required to filter things out. Due to the way the Internet works, it could be possible that the flood would not be distributed evenly among the interfaces, so that one could still be using at a fraction of capacity.

**Atack plan to compromise an internal system through the perimeter system**

Scott's system uses a Nokia firewall appliance. Runs CheckPoint Firewall-1 on top of a BSD flavor of Unix. There is no apparent bug on SecurityFocus that would allow an external user to take control of the machine. So, for the time being, a direct attack to take control of the firewall is not possible.

In order to compromise an internal system through the perimeter system, it is easier to access what we are actually permitted to do. That means, we'd better use the protocols that are actually passed through

the firewall, such as SMTP, HTTP or DNS.

The weakest link, and best choice might be the web server. There are several reasons for this:

- HTTP traffic passes through the firewall, and it is usually not checked syntactically against attacks. Because of the speed penalty, application-level firewalling is not commonly used for web.
- There are vulnerabilities that affect common Web servers (Microsoft), and there are ways, using off-the-shelf programs, to take SYSTEM control over them.
- If there is database access through the web, passwords could be stored in the server. Besides, access to the database for brute-force password attacks, denial of service is already granted due to firewall rules.
- If custom programming is used for web applications, it could be interesting to see if the application programmers have actually used safe practices to screen for illegal characters and excessive input length.

What could we do:

Test if the server is Microsoft based. We could use nmap with the -O option to determine the operating system. With any luck, it runs Microsoft Windows NT/2000 and Internet Information Server (IIS). Chances are that it might be still unpatched. Given that, we could run Nessus (a vulnerability checker) to check for common IIS vulnerabilities and look for the proper exploit to get access.

If the system does not run Microsoft, or is already patched, the next step would be to study the interface. Check all the forms, and try to explore if the system accepts weird characters. If the system is Perl-based, arguments could be passed directly to other parts due to sloppy programming.
If we get shell access finally, the next step would be to determine what to do: look at files, search for database passwords or other sensitive information, or get administrator access using vulnerabilities of the operating system used in the machine.

There is not an apparent way to get to a machine belonging to the internal network.

**List of References**

Security Focus. " Check Point Firewall-1 4.1 Denial of Service Vulnerability" 17 Jan 2001. URL: http://www.securityfocus.com/cgi-bin/vulns-item.pl?section=discussion&id=2238 (9 Oct 2001).

Security Focus. " Check Point Firewall-1 GUI Log Viewer Vulnerability " 12 Set 2001. URL: http://www.securityfocus.com/cgi-bin/vulns-item.pl?section=discussion&id=3336 (9 Oct 2001).

SecurityFocus. " Fast Mode TCP Fragment Vulnerability". 14 Dec 2000. URL: http://www.securityfocus.com/cgi-bin/vulns-item.pl?section=discussion&id=2143 (9 Oct 2001).

Ziegler, Robert. Linux Firewalls. Indianapolis: New Riders, 1999.