Global Information Assurance Certification Paper

Security Proposal for GIAC Enterprises

SANS Firewalls, Perimeter Protection, and VPNs

GCFW Practical Assignment Version 1.6a

Brian Collins

Foreword

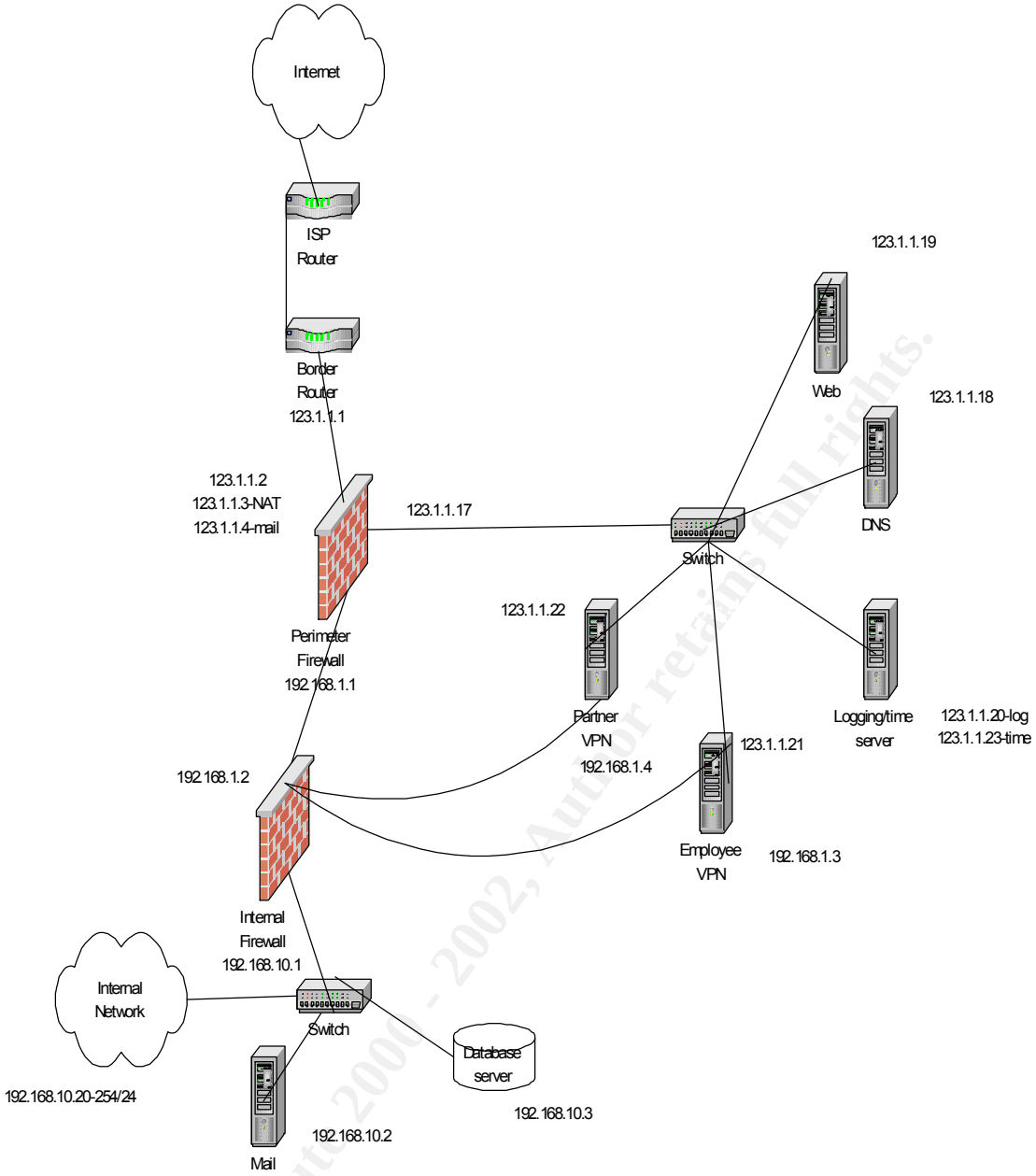
When I returned from SANS GCFW training and informed my employer of my new-found knowledge, the response was, "Great! Do some research and tell us what you think we should change (or not)." So I did. I recommended a few changes, including the installation of a newer firewall. After I made the recommendations, the new response was, "Great! When can you put all this in place?" And thus began my journey. That (yet-ongoing) project and my own endeavor to build a personal firewall for home where I'm on a cable-modem network have been very educational and have made the book learning very real and practical. The hands-on experiences of these weeks have been invaluable.

Introduction

We at Acme Network Services, Inc., have been contracted by GIAC Enterprises to design and implement a security architecture that will protect their network assets in the fortune cookie industry. What follows are the specifics of our proposed design, including what we have deemed to be the best architecture for GIAC's needs, rulesets that provide sufficient security while still providing necessary access, and a thorough audit of our design by our IT department.

Assignment 1 - Security Architecture

For the network structure and defense of GIAC Enterprises, we propose a simple architecture which will afford both flexibility and strong security. The attached diagram demonstrates in a broad stroke how we intend to implement this. To protect critical resources (web/ftp server, mail server, DNS server, logging server, database server, and our internal network) we propose the implementation of a border router, a perimeter firewall, and a VPN server. For further protection, we will install an internal firewall. This will allow us to install additional VPN servers in the future without needing host-based firewalling on each. Our diagram is on the following page.



We take into consideration for this project the following needs: customers, suppliers, partners, and internal users. All of these groups have specific needs pertaining to our network and we will accommodate those needs, as much as possible, without compromising security. What follows are more specifics of how we intend to confront these needs. Note that we speak from here on not only as Acme Network Services, but also as the IT Department of GIAC Enterprises.

I. Customer access

Our customers will purchase our fortune cookie sayings online via https. We intend to allow them to download purchased sayings by the same method. Once a purchase is complete, the messages purchased will be packaged into a compressed file which will be made available to the customer by login and password. This should not present undue effort on the part of the customer, since most purchases are made by repeat customers. Requiring them to set up a login and password with their initial

purchase works best. This method will require us to pay closer attention to the web server, since files to be downloaded will be stored on the web server itself.

II. Supplier access

We would like to take delivery of fortunes we purchase by the same means that we deliver to customers. However, we realize that not every vendor will be able to give us that capability. Therefore we will provide a system whereby suppliers may upload fortune sayings to us.

We have considered allowing VPN access to our database, but we realize that, when a vendor has many sayings to deliver, this process could be very lengthy.

We have also pondered the idea of an FTP login through a VPN connection. This would allow for single-file uploads; a more convenient solution, but also a security hole, since a security breach on the vendor side would allow an attacker FTP access to fill up our receiving server, and thereby offer downloads of warez, porn, etc. We acknowledge that this type of compromise would only happen internally with our vendors, but with so many vendors, that is still a lot of potential holes. It is perhaps not a likely scenario, but it is still more likely than we feel comfortable allowing.

For the time, the best solution seems to be allowing customers to email files to an account at GIAC. This solution is rather inefficient, but it does provide better security. For one, it cuts out any chance of an intruder accessing an incoming FTP server. Second, while it is possible for someone to use this channel to perform a DoS attack on the network (by mass-mailing the receiving account until the server's hard drive is full), it is very unlikely. Something about having a login and password seems more encouraging to bad guys than just being able to fill up a mail server anonymously. Therefore when we have customers who cannot allow us to download fortunes, we will instead ask them to email them to us as attachments in compressed format. The incoming mail server and the receiving workstation can scan these files for virus infections.

III. Partner access

To permit our partners to obtain fortunes for translation and reselling, we will grant VPN access to a server containing fortunes for retrieval. They will be given read-only accounts to a server that is past our VPN server, on our private network. This will require the running of an FTP server on our delivering server, but the fact that they will only be able to read files will make intrusion less attractive to an attacker. Such an intruder would find no haven for uploading files. We concede that this will require us to be on top of any updates for the FTP server we run.

IV. Employees

Internal users will be assigned private (non-routable) IP addresses which will be translated by NAT at the perimeter firewall. They will be granted, per management request, (almost) unrestricted outbound access to most common services of the internet, which are http/https, and FTP. Any further access required will have to pass a review process, going through department heads, IT, and (ouch) upper management. We will allow the IT department to manage the servers residing on the service network via SSH. Users who are outside the network (travelling, telecommuting, etc.) will be granted VPN access.

We have chosen a small mix of components for our proposed security architecture. In the interest of saving money, we will utilize well-configured but inexpensive and easily-manageable

products where possible. You will note our extensive use of Red Hat Linux. It is a well-supported distribution and is flexible. And while a hole found in the version we implement may well mean patching many machines quickly, the ease of management and lack of diversity will permit us to be efficient in training on and management of these machines. At the time of our proposal, the Linux-router project[i] seemed robust enough, except that, apparently lacking support for the 2.4 kernel, it prevented us from using Netfilter. A variant of that, Coyote Linux[ii], does support the 2.4 kernel and our R&D department is currently evaluating that product, especially since it is much thinner and, for a firewall, would have much less overhead. Whether the distribution proves up to the task or too limited remains to be seen. We have high hopes for it.

Border router

For our border router, we chose a Cisco model 3640. Based on past network statistics given us by GIAC's ISP, Yosemite Net, and based on future projections, the 3640 seems capable for the foreseeable future. Since a border router is the first line of defense, we will use it to filter out spoofed and unassigned public IP addresses, to drop (silently) NetBIOS traffic, and to drop undesirable ICMP traffic. However, the border router's primary job is routing, the task it specializes in, and we will not task it with very much filtering.

Perimeter firewall and internal firewall

We will custom-build a machine running Red Hat Linux 7.1 with Netfilter for the perimeter firewall and another similarly configured for the internal firewall. Though hardware is not the major focus of this proposal, we note that we chose a Tyan Tiger dual-processor AMD system for each firewall. We have chosen 2-3com 64-bit 4-port PCI Ethernet cards, for networking in the perimeter firewall, and 1 identical card for the internal firewall. This will meet our current needs and give us room to grow. The built-in 3com NICs on the motherboards will go unused for now. We considered the Tyan Tiger MP board, but scrapped that plan because of BIOS issues which we personally experienced with that board. If a later BIOS revision resolves that problem, we will reconsider the Tiger for future builds, which will allow us to avoid wasting the on-board NICs.

The perimeter firewall determines what traffic (of that which survives the border router) makes it where on the network. From the outside world, the following guidelines will be implemented. We note that state matching will be implemented at every level.

Inbound traffic from the internet

All incoming requests to ports 80 and 443 will be dropped and logged, except those destined for our web server. All incoming requests to port 25 will be dropped and logged, except those destined for our mail server; those packets will be forwarded to our mail server residing inside our network. Users who wish to check mail from the Internet (home DSL lines, dialup accounts, roving salespeople, etc.) will be required to connect to the VPN server, which will in turn grant them access to the mail server, which is behind our firewalls and has no public IP. The perimeter firewall will also permit incoming DNS (port 53).

Outbound traffic

Users on our network will be given access to the Internet as described earlier and their IP addresses NATted to a public IP assigned from our public pool. Users will send

mail through the internal mail server, which will have its IP address reassigned using one-to-one NAT at the perimeter firewall. The outbound IP will resolve to mail.giac.net, so that the mail will not appear to have come from the firewall itself.

Machines on the service network will be granted outbound access for established connections. The time server will be allowed to talk out on port 123, as will the DNS server on port 53.

Miscellaneous

All hosts on the service network will talk to the log server. The perimeter firewall and the border router will also write to the log server. The internal firewall, mail server, and database server will all log internally.

Internal traffic

The internal firewall will be configured so that the particular source IP coming from one of the VPN servers will determine what access is available. Partners will be allowed access to one VPN, and will only be granted FTP (ports 20/21) access to the read-only sites set up for them individually. Employees coming from outside will be assigned to the other VPN server and will have a different set of IP addresses; they will also get full access to the network.

The internal network will also be granted access to the service network. Internal machines will be allowed to sync time and to upload via FTP to the web server. They will also be permitted to access the DNS server for domain requests. As indicated earlier, internal machines will be allowed to manage servers on the service network via ssh.

VPN servers

The VPN servers will also run Red Hat Linux 7.1 and PPTP for the VPN server. The choice of PPTP over IPSec comes at a time when IPSec is still difficult to implement and manage at times. We will continue to research IPSec as a future improvement. It holds promise. We chose to go with PoPToP for its flexibility and wide-spread support. For hardware, again we chose the Tyan Thunder as a motherboard. We will this time utilize the on-board 3com NICs. We will also keep spare 3com 3c905B/C cards in stock in case of component failure.

Our VPN servers will reside on the service network, with another interface to the outside network of the internal firewall. The internal firewall will be used to filter packets from the VPN servers, instead of using host-based firewalling. This will permit us to add more VPN servers in the future without the need to modify the firewall greatly or do host-based firewalling on the VPNs. The VPN servers will exist to provide secure communications from partners and from employees outside our network.

Other hardware

There are other components in our design, though they are the devices protected by our defense system rather than pieces of the system itself. On our service network are: a DNS server, a web server, and a logging server also hosting time services. Behind our internal firewall is our database server (with an FTP site for partners) and a mail server.

Assignment 2 - Security Policy

With a large promise made, we now set about to fulfil it. Starting at the top, we will provide the ACLs of the Cisco border router, then move on to our perimeter firewall's rulesets and likewise the internal firewall's rulesets, and finally explore the setup of the VPN servers.

For this policy, we assume the following:

The ISP (serial) side of our router has been assigned 122.1.1.50
Our ISP has given us123.1.1.0-123.1.1.31.
We will subdivide our allocation as follows:

123.1.1.0/29 (addresses 1-7) will be our "Internet" network. We will utilize 123.1.1.1 as the inside IP of the Cisco and 123.1.1.2 as the outside of the perimeter firewall.
123.1.1.16/28 (addresses 17-31) will be our service network.
The remaining IPs (123.1.1.8/29) (addresses 9-15) we will reserve for later use.

We will utilize 192.168.1.0/28 (addresses 1-15) for our network between the two firewalls and the inside of our VPNs.
We will utilize 192.168.10.0/24 (addresses 1-254) for our internal network.

(Note by author: I acknowledge that 122.0.0.0 and 123.0.0.0 are as of yet unassigned and are reserved by ARIN and in real life would likely be dropped by any well-configured router. I use them here solely for example.)

Security policy of the Cisco 3640

We have chosen to utilize extended ACLs on the Cisco. They will be as follows.

```
Interface Serial 0
      ip address 122.1.1.50 255.255.255.255
      ip access-group 101 in
! private and otherwise unexpected IP addresses
access-list 101 deny ip 10.0.0.0 0.255.255.255 any log
access-list 101 deny ip 172.15.0.0 0.240.255.255 any log
access-list 101 deny ip 192.168.0.0 0.0.255.255 any log
access-list 101 deny ip 224.0.0.0 31.255.255.255 any log
access-list 101 deny ip 127.0.0.0 0.255.255.255 any log
access-list 101 deny ip host 0.0.0.0 any log
! our own IP allocation should not show up incoming to this
      interface
access-list 101 deny ip 123.1.1.16 16.0.0.0 any log
access-list 101 deny ip 123.1.1.8 8.0.0.0 any log
! leave your NetBIOS at the door, please
access-list 101 deny tcp any any range 135 139
access-list 101 deny udp any any range 135 139
access-list 101 deny tcp any any 445
access-list 101 deny tcp any any 445
```

```
Interface Ethernet 0
      ip address 123.1.1.0 8.0.0.0
      ip access-group 102 in
! permit through what should get through, nothing more
access-list 102 permit 123.1.1.0 8.0.0.0
access-list 102 permit 123.1.1.16 16.0.0.0
access-list 102 deny tcp any any range 135 139
access-list 102 deny udp any any range 135 139
access-list 102 deny tcp any any 445
access-list 102 deny udp any any 445
access-list 102 deny any log
```

Configuration of the Perimeter firewall

The perimeter firewall will handle the bulk of our packet filtering.
Eth0: interface facing Cisco, IP 123.1.1.2 (also 123.1.1.3 & 4)
Eth1: interface facing internal net, IP 192.168.1.1
Eth2: interface facing service net, IP 123.1.1.17
Under the folder /etc/rc.d we have the firewall script, rc.firewall. It looks as follows. In researching example firewall scripts, I learned from the experiences of others and have incorporated some of their wisdom here.

```
#!/bin/bash

# eth0 = outside interface
# eth1 = inside interface
# eth2 = service net interface
# besides standard rules (INPUT, OUTPUT, etc.) we define the following:
#  conditions first...
#
#  Note: by using ACCEPT, we imply that we silently accept a match
#  Now what do we do with matches...
#  permitlog: matches that we will permit but log (should be none or
#           (should be none or few)
#  droplog: matches that we will drop and log
#  silent: matches that we drop silently
#  scanflag: special ruleset for known scans using TCP flags
#  ping: to accept and log pings
#  badguy: folks that we have identified as persistent attackers
#  spoofed: traffic that's obviously been spoofed
#

iptables -F
iptables -F INPUT
iptables -F OUTPUT
iptables -F FORWARD
iptables -F -t nat
iptables -X scanflag
iptables -X badguy
iptables -X ping
iptables -X spoofed
iptables -X permitlog
iptables -X droplog
iptables -X silent
```

```
## Start targets
# permit and log what we send here
#  this is only here in case we want to permit some traffic but keep track of
it
#   for some reason; translated: if we get paranoid about something
iptables -N permitlog
iptables -A permitlog -m limit --limit 30/minute -j LOG --log-prefix
"Permitted:"
iptables -A permitlog -j ACCEPT

# drop and log what we send here
iptables -N droplog
iptables -A droplog -m limit --limit 30/minute -j LOG --log-prefix "Dropped:"
iptables -A droplog -j DROP

# silently drop what we send here
# Note: we would only use this for stats, to see what was not dropped by
other chains
iptables -N silent
iptables -A silent -j DROP

# isolate flag scan/DoS patterns
iptables -N scanflag
iptables -A scanflag -m limit --limit 30/minute -j LOG --log-prefix
"Scanflag:"
iptables -A scanflag -j DROP

# ping
iptables -N ping
iptables -A ping -m limit --limit 30/minute -j LOG --log-prefix "PING:"
iptables -A ping -j ACCEPT

# isolate "badguy" patterns
iptables -N badguy
iptables -A badguy -m limit --limit 30/minute -j LOG --log-prefix "BADGUY:"
iptables -A badguy -j DROP

# spoofed traffic
iptables -N spoofed
iptables -A spoofed -j LOG --log-prefix "SPOOFED:"
iptables -A spoofed -j REJECT
## End targets

# technically we don't need these, since we drop everything else
#  at the end and log it; but defining the default is wise, plus
#  if we wish not to log the rest of our dropped traffic, we can comment
#  out the lines at the end
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

# state matching; with this in place we allow established sessions
#   through, along with related sessions such as the FTP control session
#   that accompanies the data session
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
```

```
# let's NAT; mail server first, then everybody else
#  here, we give private IP traffic a public IP
iptables -t nat -A POSTROUTING -s 192.168.10.2/32 -o eth0 -j SNAT --to
123.1.1.4
iptables -t nat -A POSTROUTING -s 192.168.10.0/24 -o eth0 -j SNAT --to
123.1.1.3

# mail server
#  traffic coming to the mail server gets port-forwarded
iptables -t nat -A PREROUTING -d 123.1.1.4/32 -p tcp --dport 25 -i eth0 -j
DNAT --to 192.168.10.2:25
iptables -t nat -A PREROUTING -d 123.1.1.4/32 -p udp --dport 25 -i eth0 -j
DNAT --to 192.168.10.2:25

## Inbound rules (and by inbound we generally mean to THIS device, not the
network
#           although we do provide some generic rules here)
# anything going to the broadcast (to EVERYBODY) address cannot be good or
necessary
#
iptables -A INPUT -d 255.255.255.255 -j droplog
# link-local, private Ips
# these IPs should NEVER show up on the internet side of our firewall
iptables -A INPUT -i eth0 -s 10.0.0.0/8 -d 0/0 -j DROP
iptables -A INPUT -i eth0 -s 172.16.0.0/12 -d 0/0 -j DROP
iptables -A INPUT -i eth0 -s 192.168.0.0/16 -d 0/0 -j DROP
iptables -A INPUT -i eth0 -s 127.0.0.0/8 -d 0/0 -j DROP
iptables -A INPUT -i eth0 -s 224.0.0.0/4 -d 0/0 -j DROP
iptables -A INPUT -i eth0 -s 240.0.0.0/5 -d 0/0 -j DROP
# when a Windows (and others)machine boots with no IP assigned, it assumes
#  one in this subnet until somebody gives it one; we don't care.
iptables -A INPUT -i eth0 -s 169.254.0.0/16 -d 0/0 -j DROP
# spoofed traffic; anything showing up here probably indicates
#   a compromised machine somewhere on our network.
#  these rules deal with IP addresses showing up on an interface where
#    they should not
iptables -A INPUT -i eth0 -s 123.1.1.16/16 -d 0/0 -j spoofed
iptables -A INPUT -i eth1 -s 123.1.1.0/27 -d 0/0 -j spoofed
iptables -A INPUT -i eth2 -s 123.1.1.0/29 -d 0/0 -j spoofed
# we allow incoming ssh from machines on the private network, so they can
#   manage this firewall
iptables -A INPUT -i eth1 -s 192.168.10.0/24 -d 192.168.1.1/32 -p tcp --dport
22 -j ACCEPT
iptables -A INPUT -i eth1 -s 192.168.10.0/24 -d 192.168.1.1/32 -p udp --dport
22 -j ACCEPT

# specifically REJECT identd, tell other machine to hang-up the phone
iptables -A INPUT -i eth1 -p tcp --dport 113 -j REJECT
# flag-related scans; traffic with flags set as described below are
#  indicative of malicious traffic and attemps such as network mapping
iptables -A INPUT -p tcp --tcp-flags ALL FIN,URG,PSH -j scanflag
iptables -A INPUT -p tcp --tcp-flags ALL ALL -j scanflag
iptables -A INPUT -p tcp --tcp-flags ALL SYN,RST,ACK,FIN,URG -j scanflag
iptables -A INPUT -p tcp --tcp-flags ALL NONE -j scanflag
iptables -A INPUT -p tcp --tcp-flags SYN,RST SYN,RST -j scanflag
iptables -A INPUT -p tcp --tcp-flags SYN,FIN SYN,FIN -j scanflag
```

```
# we like some kinds of ICMP (echo reply, dest unreach,
#      time exceed, echo request) but log it all
# it may be a bad idea to allow the world to ping us, but we'll log it
#   and permit it unless we see some trouble
iptables -A INPUT -p icmp --icmp-type 0 -j ping
iptables -A INPUT -p icmp --icmp-type 3 -j ping
iptables -A INPUT -p icmp --icmp-type 11 -j ping
iptables -A INPUT -p icmp --icmp-type 8 -j ping
iptables -A INPUT -p icmp -j droplog

# NetBIOS is noisy and really has no business coming to our network
#  from the internet; because this rule specifies no interface, we are
#  basically dropping any of it destined for this device, which doesn't
#  speak NetBIOS anyway; also, the router should've caught this anyway
iptables -A INPUT -p tcp --dport 135:139 -j DROP
iptables -A INPUT -p udp --dport 135:139 -j DROP
iptables -A INPUT -p tcp --dport 445 -j DROP
iptables -A INPUT -p udp --dport 445 -j DROP

# we don't care about "icabrowser" (Winframe) or DirectPlay
# we could log this traffic, but it just gets in the way
# lots of this kind of traffic was seen in initial testing
iptables -A INPUT -p tcp --dport 47624 -j DROP
iptables -A INPUT -p udp --dport 47624 -j DROP
iptables -A INPUT -p tcp --dport 1604 -j DROP
iptables -A INPUT -p udp --dport 1604 -j DROP
#
## End of inbound rules

## Forward rules
# Generally speaking….. (rules that apply to any direction forwarding)
# drop NetBIOS here so it doesn't get logged; fills up log files quickly
# and again, we don't want NetBIOS crossing networks anyway
iptables -A FORWARD -p tcp --dport 135:139 -j DROP
iptables -A FORWARD -p udp --dport 135:139 -j DROP
iptables -A FORWARD -p tcp --dport 445 -j DROP
iptables -A FORWARD -p udp --dport 445 -j DROP
# flags, flags, flags
iptables -A FORWARD -p tcp --tcp-flags ALL FIN,URG,PSH -j scanflag
iptables -A FORWARD -p tcp --tcp-flags ALL ALL -j scanflag
iptables -A FORWARD -p tcp --tcp-flags ALL SYN,RST,ACK,FIN,URG -j scanflag
iptables -A FORWARD -p tcp --tcp-flags ALL NONE -j scanflag
iptables -A FORWARD -p tcp --tcp-flags SYN,RST SYN,RST -j scanflag
iptables -A FORWARD -p tcp --tcp-flags SYN,FIN SYN,FIN -j scanflag
# We don't take broadcasts.  But we want to know who's responsible.
iptables -A FORWARD -d 255.255.255.255 -j droplog

# we'll allow certain ICMP between service net and internal net
# for diagnostics, we'll let us ping ourselves
iptables -A FORWARD -i eth1 -d 123.1.1.16/28 -p icmp --icmp-type 0 -j ACCEPT
iptables -A FORWARD -i eth1 -d 123.1.1.16/28  -p icmp --icmp-type 3 -j ACCEPT
iptables -A FORWARD -i eth1 -d 123.1.1.16/28 -p icmp --icmp-type 11 -j ACCEPT
iptables -A FORWARD -i eth1 -d 123.1.1.16/28 -p icmp --icmp-type 8 -j ACCEPT
iptables -A FORWARD -i eth2 -d 192.168.10.0/24 -p icmp --icmp-type 0 -j
ACCEPT
```

```
iptables -A FORWARD -i eth2 -d 192.168.10.0/24 -p icmp --icmp-type 3 -j
ACCEPT
iptables -A FORWARD -i eth2 -d 192.168.10.0/24 -p icmp --icmp-type 11 -j
ACCEPT
iptables -A FORWARD -i eth2 -d 192.168.10.0/24 -p icmp --icmp-type 8 -j
ACCEPT

# we will drop and other ICMP traffic and log it
iptables -A FORWARD -p icmp -j droplog

##  Internet → service network  (and then some)
# to and from DNS, to allow internet to talk
# to our DNS server and let it talk to others
iptables -A FORWARD -i eth0 -d 123.1.1.18/32 -p udp --dport 53 -j ACCEPT
iptables -A FORWARD -i eth0 -d 123.1.1.18/32 -p tcp --dport 53 -j ACCEPT
iptables -A FORWARD -i eth2 -s 123.1.1.18/32 -p udp --dport 53 -j ACCEPT
iptables -A FORWARD -i eth2 -s 123.1.1.18/32 -p tcp --dport 53 -j ACCEPT
# we let everyone out there talk to our web server
iptables -A FORWARD -i eth0 -d 123.1.1.19/32 -p tcp --dport 80 -j ACCEPT
iptables -A FORWARD -i eth0 -d 123.1.1.19/32 -p udp --dport 80 -j ACCEPT
iptables -A FORWARD -i eth0 -d 123.1.1.19/32 -p tcp --dport 443 -j ACCEPT
iptables -A FORWARD -i eth0 -d 123.1.1.19/32 -p udp --dport 443 -j ACCEPT

# from router and firewall to syslog server (yes, we cheated and stuck in an
OUT rule)
# this allows our router and firewall to log to the log server
iptables -A FORWARD -i eth0 -s 123.1.1.1/32 -d 123.1.1.20/32 -p udp --dport
514 -j ACCEPT
iptables -A OUTPUT -o eth2 -s 123.1.1.17/32 -d 123.1.1.20/32 -p udp --dport
514 -j ACCEPT
# our time server has to go out and get time
iptables -A FORWARD -i eth0 -d 123.1.1.23/32 -p udp --dport 123 -j ACCEPT
iptables -A FORWARD -i eth0 -d 123.1.1.23/32 -p tcp --dport 123 -j ACCEPT
iptables -A FORWARD -i eth2 -s 123.1.1.23/32 -p udp --dport 123 -j ACCEPT
iptables -A FORWARD -i eth2 -s 123.1.1.23/32 -p tcp --dport 123 -j ACCEPT
# VPN for employees
iptables -A FORWARD -i eth0 -d 123.1.1.21/32 -p tcp --dport 1723 -j ACCEPT
iptables -A FORWARD -i eth0 -d 123.1.1.21/32 -p 47 -j ACCEPT
# VPN for partners
#  when we have a partner who needs VPN, copy the following lines and replace
#    the a.b.c.d with their network addresses
#iptables -A FORWARD -i eth0 -s aaa.bbb.ccc.ddd -d 123.1.1.21/32 -p tcp --
dport 1723 -j ACCEPT
#iptables -A FORWARD -i eth0 -s aaa.bbb.ccc.ddd -d 123.1.1.21/32 -p 47 -j
ACCEPT

## internal network → service network
# let our folks talk to the DNS
iptables -A FORWARD -i eth1 -s 192.168.10.0/24 -d 123.1.1.18 -p udp --dport
53 -j ACCEPT
iptables -A FORWARD -i eth1 -s 192.168.10.0/24 -d 123.1.1.18 -p tcp --dport
53 -j ACCEPT
# and to the web server, to ftp files to and from it as well as to browse to
it
iptables -A FORWARD -i eth1 -s 192.168.10.0/24 -d 123.1.1.19 -p tcp --dport
80 -j ACCEPT
```

```
iptables -A FORWARD -i eth1 -s 192.168.10.0/24 -d 123.1.1.19 -p udp --dport
80 -j ACCEPT
iptables -A FORWARD -i eth1 -s 192.168.10.0/24 -d 123.1.1.19 -p tcp --dport
21 -j ACCEPT
iptables -A FORWARD -i eth1 -s 192.168.10.0/24 -d 123.1.1.19 -p udp --dport
21 -j ACCEPT
iptables -A FORWARD -i eth1 -s 192.168.10.0/24 -d 123.1.1.19 -p tcp --dport
20 -j ACCEPT
iptables -A FORWARD -i eth1 -s 192.168.10.0/24 -d 123.1.1.19 -p udp --dport
20 -j ACCEPT
# let them sync time from the time server
iptables -A FORWARD -i eth1 -s 192.168.10.0/24 -d 123.1.1.23 -p tcp --dport
123 -j ACCEPT
iptables -A FORWARD -i eth1 -s 192.168.10.0/24 -d 123.1.1.23 -p udp --dport
123 -j ACCEPT
# ssh for management of machines
iptables –A FORWARD –i eth1 –s 192.168.10.0/24 –d 123.1.1.16/28 –p udp --
dport 22 –j ACCEPT
iptables –A FORWARD –i eth1 –s 192.168.10.0/24 –d 123.1.1.16/28 –p tcp --
dport 22 –j ACCEPT

## internal network → internet
#  these rules are very strict, and we foresee management recommending
#   that we "loosen things up" shortly after this is implemented;
#   but for now, we're in charge  :-)
# web surfing
iptables -A FORWARD -o eth0 -p tcp --dport 80 -j ACCEPT
iptables -A FORWARD -o eth0 -p udp --dport 80 -j ACCEPT
iptables -A FORWARD -o eth0 -p tcp --dport 443 -j ACCEPT
iptables -A FORWARD -o eth0 -p udp --dport 443 -j ACCEPT
# FTP
iptables -A FORWARD -o eth0 -p tcp --dport 20 -j ACCEPT
iptables -A FORWARD -o eth0 -p udp --dport 20 -j ACCEPT
iptables -A FORWARD -o eth0 -p tcp --dport 21 -j ACCEPT
iptables -A FORWARD -o eth0 -p udp --dport 21 -j ACCEPT

# allow the mail server to send its mail
iptables -A FORWARD -o eth0 -s 192.168.10.2/32 -p tcp --dport 25 -j ACCEPT
iptables -A FORWARD -o eth0 -s 192.168.10.2/32 -p udp --dport 25 -j ACCEPT

#
## end of Forward rules

## Rules for traffic local to this box
# localhost has to operate
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
## End of local traffic rules

## Outbound rules
#  If we want to do diagnostics from this firewall, uncomment these
#iptables -A OUTPUT -p icmp --icmp-type 0 -j ACCEPT
#iptables -A OUTPUT -p icmp --icmp-type 3 -j ACCEPT
#iptables -A OUTPUT -p icmp --icmp-type 11 -j ACCEPT
#iptables -A OUTPUT -p icmp --icmp-type 8 -j ACCEPT
#iptables -A OUTPUT -p icmp -j droplog
```

```
## Catch-all rules
# last rules; permit anything except spoofed, etc.
#  drop and log everything else; also, reject identd

iptables -A INPUT -j LOG --log-prefix "EndDropI:"
iptables -A FORWARD -j LOG --log-prefix "EndDropF:"
iptables -A OUTPUT -j LOG --log-prefix "Compromise:"
#
## End of all rulesets
```

See notes at the end of assignment 2 for sources used in determining optimal iptables configuration.

Configuration of the Internal firewall

The configuration of the internal firewall is pretty simple,so we won't go into it much here. We would like to point out how we will isolate traffic from a partner network going through the partner VPN server. Assume eth0 is the card facing the VPN and firewall, and eth1 is the card facing the internal network. We also have state-matching on this machine.

```
## don't trust this machine very far...
#
iptables -A FORWARD -i eth0 -s 123.1.1.22/32 -d 192.168.10.248/29 -p tcp --dport
20 -j ACCEPT
iptables -A FORWARD -i eth0 -s 123.1.1.22/32 -d 192.168.10.248/29 -p udp --dport
20 -j ACCEPT
iptables -A FORWARD -i eth0 -s 123.1.1.22/32 -d 192.168.10.3/32 -p tcp --dport 21
-j ACCEPT
iptables -A FORWARD -i eth0 -s 123.1.1.22/32 -d 192.168.10.3/32 -p udp --dport 21
-j ACCEPT
iptables -A FORWARD -i eth0 -s 123.1.1.22/32 -d 0/0 -j droplog
```

There are many ways to approach the internal firewall configuration. It is conceivable that we could provide some of the same protection that the perimeter firewall provides, except that we would be coming from the other direction. In other words, we could have rules that only allow port 22 to go to the service network, or that define how the internal network can talk to the service network and on what ports. This redundancy is not necessarily bad and may well be desirable.

Configuration of the VPN servers

We have chosen to utilize PPTP (Poptop v1.0.1) for VPN services. Once the kernel has been updated to 2.4.15 (learned from research at http://lists.schulte.org/pipermail/pptp-server/), we install the PPTPD software and configure it. One of the first configuration files is '/etc/pptpd.conf' and it looks as follows:

```
################################################################################
#
# GIAC vpn configuration file
#
# for PoPToP version 1.0.1
```

```
#
################################################################################

# not that this really matters when you're not using dialup
speed 115200

# where is my options file?
option /etc/ppp/options.pptp

debug

localip 123.1.1.21

# remote IP range
remoteip 192.168.1.241-253

# only listen on outside interface, prevent tunnels inside tunnels
listen 123.1.1.21
```

Furthermore, under the folder /etc/ppp we have two critical configuration files, namely chap-secrets and options.pptp. Kernel patches allow for the stripping of Microsoft domain information, thus eliminating prior headaches in using CHAP, where a computer's peer-to-peer domain (or workgroup) information would be stuck in front of the login name and requiring a separate entry in chap-secrets for every user/machine combination. To be brief, a user who might log in using two different machines from home would have to have an entry in chap-secrets for both.

Here are the contents of options.pptp:

```
proxyarp
lock
auth
debug
name GIACVPNServer1
nodefaultroute
ms-dns 123.1.1.18
require-chap
+chapms
+chapms-v2
mppe-40
mppe-128
ipcp-accept-local
ipcp-accept-remote
chapms-strip-domain
```

And from chap-secrets:

```
# Secrets for authentication using CHAP
# client        server   secret                  IP addresses
fred        *     *                  192.168.1.241
barney            *     *                  192.168.1.242
wilma       *     *                  192.168.1.243
betty       *     *                  192.168.1.244
```

Finally, we need an ipchains or iptables rule to forward packets through the machine. That would look something like this:

```
iptables -A FORWARD -s 192.168.1.0/255.255.255.0 -d
192.168.1.0/255.255.255.0 -j ACCEPT
```

We must also enable forwarding in the system by a command like this in /etc/rc.d/rc.local:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

And we enable the forwarding of proxy-arp similarly:

```
echo 1 > /proc/sys/net/ipv4/conf/all/proxy_arp
```

This concludes the demonstrations of the ACLs in our architecture.

Tutorial for implementing the perimeter firewall

The perimeter firewall will be an install of Red Hat Linux 7.1 where "Custom" was chosen during the install process. Only the most basic of elements will be chosen during install, and we will choose "no firewall," since we will configure our own settings for the firewall. After install, we will install the kernel headers and source for the most recent kernel made available by Red Hat, currently 2.4.9-12. We reconfigure the kernel (using make menuconfig) before recompiling.

Kernel recompiling takes place as follows:

```
cd /usr/src/linux-2.4
make mrproper
cp -p configs/kernel-2.4.9-athlon-smp.config arch/i386/defconfig
make menuconfig
      (output of this, in .config, has been included in the appendix
since it is so long)
      (edit Makefile to change the kernel version to 2.4.9-12smp)
make dep bzImage 2>errors
make modules 2>errors
make modules_install
mkinitrd /boot/initrd-2.4.9-12smp 2.4.9-12smp
```

Next we edit /etc/lilo.conf to tell the system to boot from the new kernel, then run `lilo -v -v`, check for errors, and if all looks well, reboot.

The file /etc/lilo.conf might look as follows:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
message=/boot/message
linear
default=smp

image=/boot/vmlinuz-2.4.2-2
        label=orig
        read-only
```

```
        root=/dev/hda5
image=/boot/vmlinuz-2.4.9-12
        label=safe
        read-only
        root=/dev/hda5
        initrd=/boot/initrd-2.4.9-12
image=/boot/vmlinuz-2.4.9-12.smp
        label=smp
        read-only
        root=/dev/hda5
        initrd=/boot/initrd-2.4.9-12.smp
```

Note: the initrd is generally useful only when you have a SCSI subsystem. We include it here for reference.

For further security on the firewall itself, we turn off all unnecessary services. The easiest way is to prevent such things from starting at boot time.

```
chkconfig --level 2345 apmd off
chkconfig --level 2345 gpm off
chkconfig --level 2345 xinetd off
chkconfig --level 2345 lpd off
```

Do this for all services you do not want running at boot time. To see which are running, run: `chkconfig --list | grep on`. If we don't need it, we don't start it. We also need to turn off ipchains (by the same method) so that iptables will run.

Once we have these things tackled, we begin the configuration of the iptables themselves. The actual rulesets, included in the first part of this section, will be included in and thus called from /etc/rc.d/rc.firewall, which itself will be called from /etc/rc.d/rc.local. We also include a script to clean out the rules, just in case we get happy and accidentally tighten down the machine too much. It merely flushes the built-in chains, and flushes and deletes the user-defined chains. Of course, if we lock ourselves out of remote access by our tinkering, we'll have to go to the console.

To avoid repeating the lengthy rules here, we refer the reader to the actual rulesets for the descriptions of each rule and their justifications. We do, however, offer a syntax explanation and the application of the rules in general.

Syntax of an iptables rule

A. adding a rule

```
iptables -A <chain> -s <source ip or net> -d <dest ip or net> -p
<protocol> --sport <source port> --dport <dest port> -j <jump to
target>
```

chain references the chain to add this rule to. It can refer to the built-in chains or a user-defined chain. Built-in chains are typically INPUT for traffic coming into (destined for but not through) this box, OUTPUT for traffic originating from this box, and FORWARD for traffic being routed through this box. A user-defined chain could be employed to group kinds of traffic and can also be a target for *-j*.

source ip references a host or a network*:*
192.168.1.0/24 would reference a whole network from 192.168.1.1 through 192.168.1.254
192.168.1.25/32 would reference only the ip 192.168.1.25
likewise, the *destination ip* has the same format

protocol references the protocol to be analyzed. Valid options are tcp, udp, icmp, or even a protocol number, such as 47 used in VPN.

source port and destination port reference the ports specified in the packet, and apply primarily to tcp and udp. They can also specify a range, such as 135:139 to reference NetBIOS.

jump to target references "what to do from here." This is where say what happens next. The most common options are ACCEPT, DROP, REJECT, and LOG. We may also jump to user-defined chains, so that we may, for example, jump to a chain that logs the traffic and then drops it. If we choose to log traffic, we may specify a prefix to stick in the log file with *--log-prefix "Text:"* after *-j LOG*. ACCEPT allows the packet to pass, DROP acts as if the packet never existed, and REJECT by default sends an ICMP message back to the host saying the port is unreachable. This is useful when we are hit with an identd request, because the REJECT will tell the host we aren't talking identd and the host will quit asking; otherwise the host may keep waiting until it times out, which would be unfriendly toward that host. The POSTROUTING chain specifies what happens after a packet has been processed for routing, and is useful with NAT to change the source IP address to something acceptable on the Internet. Conversely, the PREROUTING chain generally changes the destination IP address and/or port before we route the packet.

Another option we employ is *-t* to specify which table (by default, the system assumes we refer to the *filter* table. We may also specify the *nat* table, for translating IP addresses to other IP addresses for outbound or inbound traffic. This helps us be able to use private IPs behind the firewall while our outbound traffic appears as a public IP.

B. flushing a rule

```
iptables -F <chain>
```

With this command, we can flush a chain, which basically deletes all the rules we stuck in that chain. For example, flushing the forward chain with *ipchains -f FORWARD* will leave us with no rules for forwarding except the default.

C. setting a default policy for a chain

```
iptables -P <chain> <policy>
```

This command sets the default policy for a chain. So if we specify *iptables -P FORWARD DROP* then we are saying, "If you run the packet through all the rules and it's still not been handled, drop it." This has the same effect as placing a rule at the end of the ruleset which catches the leftovers.

D. deleting a user-defined chain

```
iptables -X <chain>
```

We use this command to delete a chain we defined.

E. defining a new chain

```
iptables -N chain
```

We use this chain to define a new chain, which we can then populate with rules using the add command.

F. getting the stats from our rules

```
iptables -L -n -v
```

This combination lists (*-L)* the current rules, does not resolve IP addresses to names (*-n)*, and gives us statistics (*-v for verbose)*. From this we can view statistics of what our filters are catching and re-arrange the rules to be more efficient. We may also zero the counters with *-Z*.

With this in mind, let us analyze 3 rules from our policy.

Sample 1:

```
iptables -t nat -A POSTROUTING -s 192.168.10.0/24 -o eth0 -j SNAT --to
123.1.1.3
```

With this rule, we use our NAT table to modify the source address of any traffic coming from class C network 192.168.10.0 (octet D being one of 1-254) to be 123.1.1.3, provided it is destined to go out interface *eth0*. To test this rule, we could run tcpdump on a machine on the other side of the firewall from the source (say, 192.168.10.5). Provided our rules allow, we could ping the outside machine from the source. The tcpdump output would show us the source IP address of the packet, which should show up as 123.1.1.3. We could further test by pinging without this NAT rule in place, and we should then see 192.168.10.5 as the source IP.

Sample 2:

```
iptables -A OUTPUT -j LOG --log-prefix "Compromise:"
```

Next we see a rule for our OUTPUT chain that writes “Compromise:” to the log file for anything matching it. At this point in the rulesets, all other chains have been tried and no match has been found. The implication is that all traffic that originates from this box and is legitimate has already been handled by other OUTPUT rules, and this traffic, therefore, likely indicates a compromise of our perimeter firewall. It may also indicate that an administrator logged into the box and wanted to do something like ftp a file from another server, and forgot that we don’t do that on our perimeter firewall. As Christ Brenton mentioned in the GCFW class, “There’s a technical term for that; it’s called a bad thing.” It doesn’t hurt to catch the “bad things” we do accidentally. The easiest way to test this rule would be to log in (either on the console or via ssh, since permit both), and try to retrieve a file via ftp from our web server. There is a FORWARD rule that permits ftp to the web server, but no OUTPUT rule. Therefore this attempt should be denied and logged. We also wish to point out that, if this box were

compromised, this is a good argument for logging on something other than this box.

Sample 3:

```
iptables -N ping
iptables -A ping -m limit --limit 30/minute -j LOG --log-prefix
"PING:"
iptables -A ping -j ACCEPT
```

Here, we define a chains (ping), then add 2 rules to it. First, we log everything that gets sent to this target, using the prefix "PING:" in the log file for easy "grepping." In doing so, we limit the number of entries to 30 per minute, so we do not overload the logging daemon on our logger. Second we accept the ping traffic. This rule is helpful for identifying the source of ping requests if we decide to permit them. To test this, we make sure the INPUT chain specifies to jump to the *ping* target when we receive echo request packets via the ICMP protocol, then we ping from a machine on the same side as the interface specified in that rule. We should see "PING:" show up in the log for each packet received.

Summary of rulesets

In general, we give access to services that we have to give access to, and block the rest. It would likely be safe for us merely to turn off unwanted services on critical machines, but this takes it one step further in that, if a machine in the DMZ (such as the web server) is compromised so that a hacker runs a telnet daemon on it, he still cannot allow access from the internet unless he does so on port 80. Moreover, since we only allow our internal network to access ftp on the web server, the internal network is basically the only place we have to worry about an attacker trying to exploit the hole of the week in ftp.

Are there still holes? Very likely so. New vulnerabilities are discovered quite often in services and operating systems. But by running some of the more secure products available, and by utilizing the security tools we have available to the best of our ability, we can cut our risks to a minimum.

In researching our methods, theory, and approach, several websites were helpful, especially as it pertains to understanding and implementing iptables. Those documents most helpful were:

http://www.linuxguruz.org/iptables/scripts/rc.firewall_004.txt
http://www.frognet.net/~aalug/docs/iptables/node15.html
http://poptop.lineo.com
http://www.neohapsis.com/neolabs/neo-ports/ (ports list)
http://www.linuxdoc.org/HOWTO/Adv-Routing-HOWTO-5.html
http://vibrationresearch.com/pptpd/example.html
http://netfilter.samba.org

(Author's note: I also referred often to the GCFW course literature, my own notes, and several man pages. And of course, the experience of building my own home firewall and testing it has helped greatly).

Assignment 3: Auditing GIAC Enterprises' new security architechture

While the thought of an audit of our firewall might have us anxious for fear that we missed something along the way, that is exactly why we should do one. Better that we should poke at it and break it than someone with less helpful intentions. And although we have tried to be careful and meticulous in the design and implementation, in truth we welcome the opportunity to see just how well we did.

Step 1: Planning our audit

To conduct our audit, we intend to have on hand 3 systems administrators, who cost us $20/hour each. One will utilize the attacking machine, another the destination machine, and the other will monitor firewall logs during the process. The victim will run tcpdump to see how the attack comes to him. On the firewall itself we will also run tcpdump. It will run knowing who the perpetrator and the intended victim are. The attacker will utilize telnet for basic attempts and nmap for more complicated attempts. We expect the audit to consume the better part of a work-day.

The best time for us to conduct this is during low-activity hours, but also during reasonable technician-awareness hours (translated: they're not sleepy). GIAC has determined to be closed the 3 days after Christmas 2001, and that is when we have chosen to do the attack.

It is entirely conceivable that we will break something during this audit. We hope not to do so, but we could. We may flood a server to sleep. We might also, though, bring such a DoS against our firewall that it stops firewalling and becomes a happy packet-forwarder with no morals. That is another good reason for the off hour audit.

Step 2: Audit day

Our first tests indicate a typo in the firewall script, where we used a mask of 255.255.0.0 (/16) instead of the correct 255.255.255.240 (/28) in a rule. We discovered this by attempting to ping the web server from a machine outside the firewall. The 'spoofed' rule had this error, and was flagging traffic as spoofed which was not. We corrected this easily and quickly.

Next we verified rules pertaining to the web server. First, we tried what should be allowed: port 80 (no https pages loaded yet, but we know that if port 80 works, identical rules for port 443 should as well). Our outside machine used lynx to open the default page on the web server, and it opened. As expected, the firewall logged none of it and dropped none of it. The tcpdump was run on eth2, which faces the service network. Here is the output of tcpdump (note that, in log entries, tcpdump notation, and nmap output, we have referred to 'scriptkiddie.somedsl.net' as our perpetrator instead of referencing an IP address):

```
      11:19:07.999914 > scriptkiddie.somedsl.net.1036 > 123.1.1.19.http: S
2190490741:2190490741(0) win 5840 <mss 1460,sackOK,timestamp 227812
0,nop,wscale 0> (DF)
      11:19:08.000272 < 123.1.1.19.http > scriptkiddie.somedsl.net.1036: S
1813148373:1813148373(0) ack 2190490742 win 5792 <mss 1460,sackOK,timestamp
112510481 227812,nop,wscale 0> (DF)
```

```
      11:19:08.000566 > scriptkiddie.somedsl.net.1036 > 123.1.1.19.http: .
1:1(0) ack 1 win 5840 <nop,nop,timestamp 227812 112510481> (DF)
      11:19:08.002016 > scriptkiddie.somedsl.net.1036 > 123.1.1.19.http: P
1:565(564) ack 1 win 5840 <nop,nop,timestamp 227812 112510481> (DF)
      11:19:08.002786 < 123.1.1.19.http > scriptkiddie.somedsl.net.1036: .
1:1(0) ack 565 win 6768 <nop,nop,timestamp 112510481 227812> (DF)
      11:19:08.006771 < 123.1.1.19.http > scriptkiddie.somedsl.net.1036: .
1:1449(1448) ack 565 win 6768 <nop,nop,timestamp 112510482 227812> (DF)
      11:19:08.008003 < 123.1.1.19.http > scriptkiddie.somedsl.net.1036: .
1449:2897(1448) ack 565 win 6768 <nop,nop,timestamp 112510482 227812> (DF)
      11:19:08.008285 < 123.1.1.19.http > scriptkiddie.somedsl.net.1036: P
2897:3156(259) ack 565 win 6768 <nop,nop,timestamp 112510482 227812> (DF)
      11:19:08.008359 < 123.1.1.19.http > scriptkiddie.somedsl.net.1036: F
3156:3156(0) ack 565 win 6768 <nop,nop,timestamp 112510482 227812> (DF)
      11:19:08.009713 > scriptkiddie.somedsl.net.1036 > 123.1.1.19.http: .
565:565(0) ack 1449 win 8688 <nop,nop,timestamp 227813 112510482> (DF)
      11:19:08.009844 > scriptkiddie.somedsl.net.1036 > 123.1.1.19.http: .
565:565(0) ack 2897 win 11584 <nop,nop,timestamp 227813 112510482> (DF)
      11:19:08.009919 > scriptkiddie.somedsl.net.1036 > 123.1.1.19.http: .
565:565(0) ack 3156 win 14480 <nop,nop,timestamp 227813 112510482> (DF)
      11:19:08.018643 > scriptkiddie.somedsl.net.1036 > 123.1.1.19.http: F
565:565(0) ack 3157 win 14480 <nop,nop,timestamp 227814 112510482> (DF)
      11:19:08.018894 < 123.1.1.19.http > scriptkiddie.somedsl.net.1036: .
3157:3157(0) ack 566 win 6768 <nop,nop,timestamp 112510483 227814> (DF)
```

The first packet shows the SYN bit set with traffic going to the web server's http port, then we see the server responding with a SYN/ACK and the client responding with the completion of the handshake in an ACK. After that is the indication of the server sending data to the client.

Next we see the results of an attempt to FTP to the server from beyond the firewall:

```
11:27:15.389985 < scriptkiddie.somedsl.net.1041 > 123.1.1.19.ftp: S
2724506779:2724506779(0) win 5840 <mss 1460,sackOK,timestamp 276563
0,nop,wscale 0> (DF)
11:27:18.388844 < scriptkiddie.somedsl.net.1041 > 123.1.1.19.ftp: S
2724506779:2724506779(0) win 5840 <mss 1460,sackOK,timestamp 276863
0,nop,wscale 0> (DF)
```

The attacker tries twice, but never sees a packet. We used tcpdump this time on eth0, facing the outside world, since we did not anticipate the packets making it through the firewall. Surely enough, listening on eth2 yielded no packets. The firewall entries looked like this:

```
Dec 26 11:27:15 t253 kernel: EndDropF:IN=eth0 OUT=eth2
SRC=scriptkiddie.somedsl.net DST=123.1.1.19 LEN=60 TOS=0x00 PREC=0x00 TTL=63
ID=38088 DF PROTO=TCP SPT=1041 DPT=21 WINDOW=5840 RES=0x00 SYN URGP=0
Dec 26 11:27:18 t253 kernel: EndDropF:IN=eth0 OUT=eth2
SRC=scriptkiddie.somedsl.net DST=123.1.1.19 LEN=60 TOS=0x00 PREC=0x00 TTL=63
ID=38089 DF PROTO=TCP SPT=1041 DPT=21 WINDOW=5840 RES=0x00 SYN URGP=0
```

The chain catching this was our "EndDropF" chain, indicating it was the end chain which drops undesirable forward (routed) traffic. Telnet yielded similar results, with the DPT being 23 of course. Telnet also yielded similar tcpdump output.

All these attemps indicate things work OK on the surface, but we need to go a little deeper and run something like nmap. So, using the following command:

```
nmap -v 123.1.1.19 -oN webmap.log
```

we get the following output in the file webmap.log:

```
# nmap (V. 2.54BETA22) scan initiated Wed Dec 26 13:15:08 2001 as: nmap -v -
oN webmap.log 123.1.1.19
Interesting ports on  (123.1.1.19):
(The 1540 ports scanned but not shown below are in state: filtered)
Port       State       Service
80/tcp     open        http
443/tcp    closed      https


# Nmap run completed at Wed Dec 26 13:17:53 2001 -- 1 IP address (1 host up)
scanned in 165 seconds
```

This indicates that our firewall is allowing ports 80 and 443 through (though nothing shows up on port 443 yet), and filtering the rest. This is what we expected. Firewall entries during the probe were much like these two:

```
Dec 26 13:51:37 t253 kernel: EndDropF:IN=eth0 OUT=eth2
SRC=scriptkiddie.somedsl.net DST=123.1.1.19 LEN=60 TOS=0x00 PREC=0x00 TTL=63
ID=56458 DF PROTO=TCP SPT=1227 DPT=156 WINDOW=5840 RES=0x00 SYN URGP=0
Dec 26 13:51:37 t253 kernel: EndDropF:IN=eth0 OUT=eth2
SRC=scriptkiddie.somedsl.net DST=123.1.1.19 LEN=60 TOS=0x00 PREC=0x00 TTL=63
ID=2394 DF PROTO=TCP SPT=1228 DPT=707 WINDOW=5840 RES=0x00 SYN URGP=0.
```

These entries correlate with nmap's output. An ACK scan gave the same results, telling us that ports 80 and 443 are unfiltered. Everything else went bye-bye and got logged as a flag scan, with tcpdump on the victim machine never receiving the packets.

Moving on to the logging server, we want to be able to connect to his syslog port (udp 514) and nothing else. We modify /etc/rc.d/init.d/syslog to make syslogd accept remote entries to his log file. Then we modify the router and the firewall to log to him. Log entries show up in the file, so we know he accepts connections. Since syslogd uses UDP and doesn't respond, we expect nmap to tell us everything is either closed or filtered, or perhaps even that the host is down.

Running nmap straight out (nmap -v 123.1.1.20) returns the message that the host is likely down, with the recommendation to add the -P0 option, which disables pinging before scanning. So we do.

```
Starting nmap V. 2.54BETA22 ( www.insecure.org/nmap/ )
No tcp,udp, or ICMP scantype specified, assuming vanilla tcp connect() scan.
Use -sP if you really don't want to portscan (and just want to see what hosts
are up).
Host  (123.1.1.20) appears to be up ... good.
Initiating Connect() Scan against  (123.1.1.20)
The Connect() Scan took 1664 seconds to scan 1542 ports.
All 1542 scanned ports on  (123.1.1.20) are: filtered

Nmap run completed -- 1 IP address (1 host up) scanned in 1664 seconds
```

The same server hosts time services (port 123) on a different IP address, 123.1.1.23. Like syslogd, this uses UDP and does not respond to the TCP scan.

We focus now on the DNS server. Running nmap against it shows us only port 53 being available, and everything else being filtered.

Initially, all connections to port 25 on 123.1.1.4 (the public IP of the mail server) initially failed. Obtaining a clue from the firewall logs, understand we need to add the following lines to rc.firewall, and we place them right after the DNAT lines.

```
iptables -A FORWARD -i eth0 -d 192.168.10.2/32 -p tcp --dport 25 -j ACCEPT
iptables -A FORWARD -i eth0 -d 192.168.10.2/32 -p udp --dport 25 -j ACCEPT
```

Now attempting to connect to port 25 on the public IP for the mail server yields:

```
220 mail.giac.net ESMTP Sendmail 8.11.6/8.11.6; Wed, 26 Dec 2001 16:07:59 -
0500
```

and leads us to conclude that our port forwarding works.

We should also run nmap against the mail IP address. We do so and get:

```
Host  (123.1.1.4) appears to be up ... good.
Initiating Connect() Scan against  (123.1.1.4)
Adding TCP port 25 (state open).
The Connect() Scan took 2 seconds to scan 1542 ports.
Interesting ports on  (123.1.1.4):
(The 1534 ports scanned but not shown below are in state: closed)
Port       State       Service
25/tcp     open        smtp
135/tcp    filtered    loc-srv
136/tcp    filtered    profile
137/tcp    filtered    netbios-ns
138/tcp    filtered    netbios-dgm
139/tcp    filtered    netbios-ssn
445/tcp    filtered    microsoft-ds
```

We anticipate this because the IP 123.1.1.4 is actually the firewall itself (except port 25, which gets rightfully forwarded) and we are certainly filtering those ports mentioned above.

With the VPN servers not yet in place, we will have to test those later.

Finally, we want to run nmap against the firewall itself. The TCP scan shows the same results we got against 123.1.1.4. A scan using UDP got too many drops and prompted us to give up. Other scans (XMAS, Null, FIN) gave mixed results. The XMAS scan gave nothing, while the FIN scan showed the NetBIOS ports open, though they are not. Changing the rule to REJECT those ports rather than DROP gives us the answer that they are filtered. We presume, then, that nmap, upon not getting a response to the NetBIOS FIN probes, assumes they are open. The NULL scan determined that everything was closed.

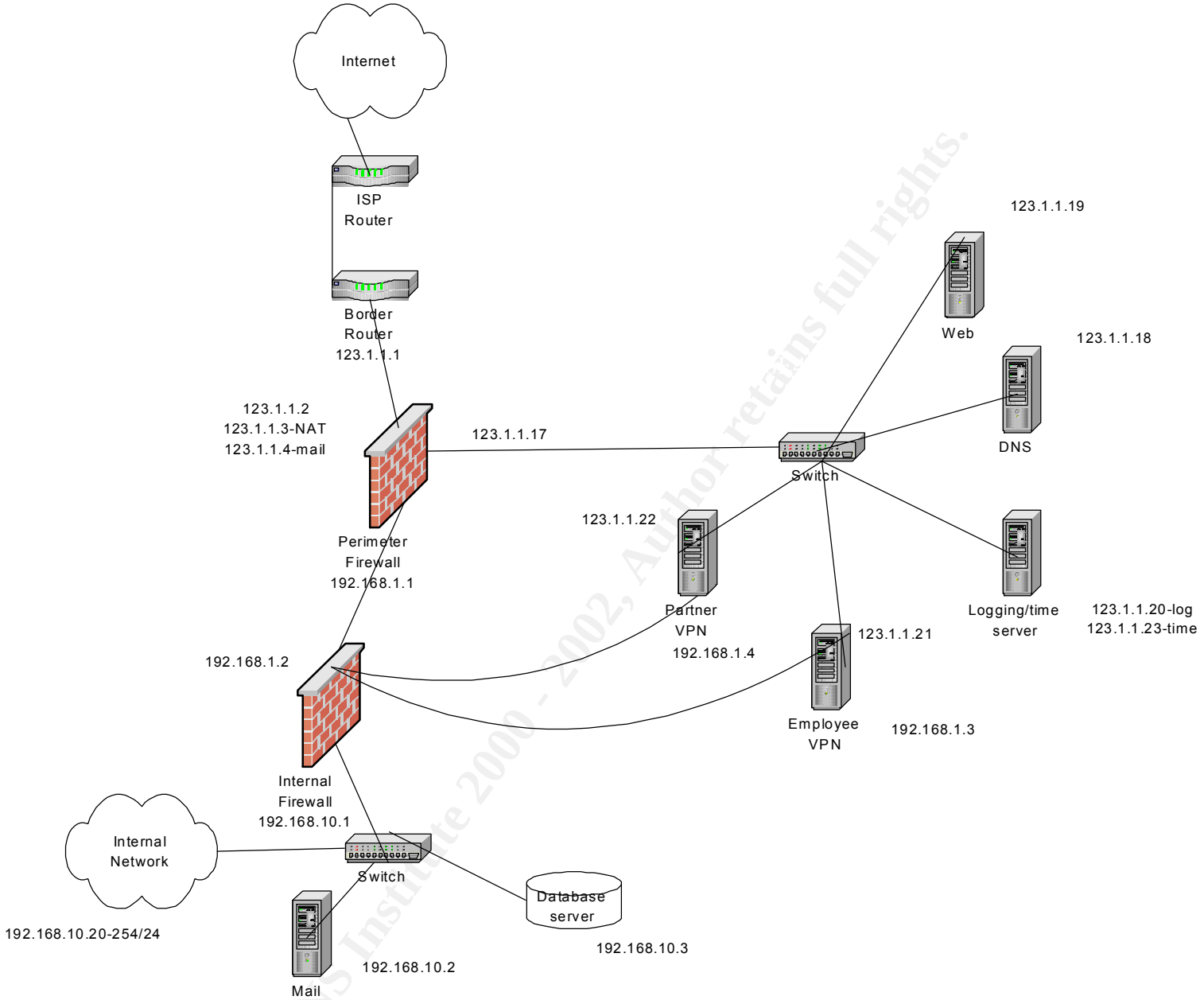
We intend to repeat the attacks from inside the firewall and from the service network. Since many of the rulesets are identical with the exception of destination networks/IP

addresses and incoming/outgoing interfaces, we expect very similar results when we perform the audit from the employee network and from the DMZ.

Step 3: What did we learn?

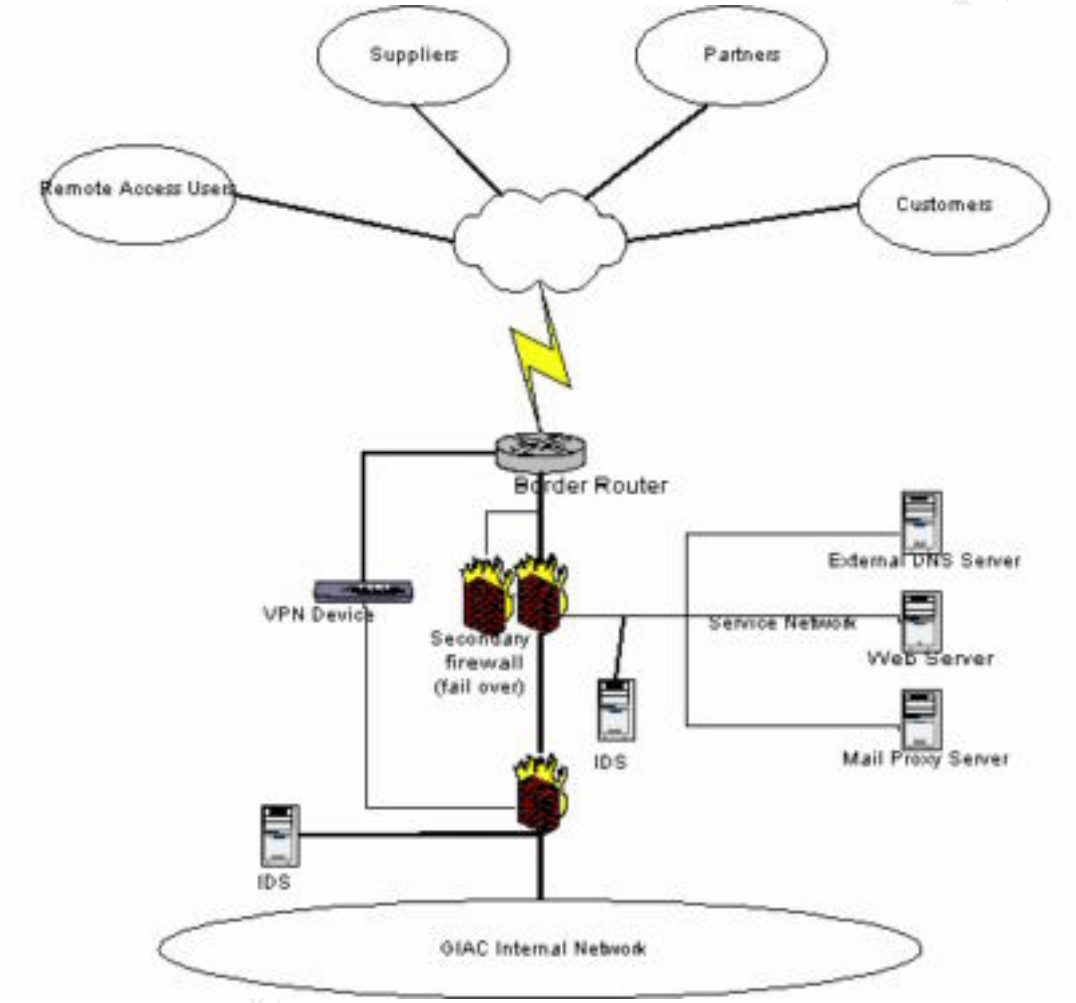
Our audit proved helpful, for we needed to correct a typographical error and fix port forwarding for our mail server. We also modified the NetBIOS rules to reject that traffic rather than drop it. The design seems to be a sturdy one and we will stick with it. We do think it wise to add intrusion detection utilizing a tool like Tripwire. Tripwire could be installed on each firewall with relative ease. It could also be installed on each of our servers in the DMZ for added protection.

After much consideration, we have also concluded that we had rather get complaints from outsiders than to allow ping traffic to cross or hit our firewall. Therefore, we also recommend removing the rulesets which permit pings to and through the firewall from the outside world, and retain the ICMP packets that are conducive to good operation (unreachables, etc.). A ping flood could theoretically disable the firewall to any extent. The mildest case would be that it simply dies and we have no access in either direction. The worst is that, as stated earlier, it simply becomes a dumb router, passing gleefully all the traffic that comes to it for routing. We repeat our design below.



Assignment 4: Design Under Fire

For assignment 4 in this practical, I chose to evaluate the design of Donna Dance-Masgay, whose approved practical I found at http://www.giac.org/practical/Donna_Dance-Masgay_GCFW.zip. Her self-audited design looks like this:



We understand that she is running Checkpoint's Firewall-1, v4.1, SP3, on a Solaris box. We have the following options available for our attack, and will choose 2 of them in an attempt to prove her design's integrity.

1. Attack the firewall itself. We could attack not only Firewall-1, but we could also see what's available to overthrow Solaris.
2. Implement DoS attacks utilizing certain floods such as TCP SYN, UDP, or ICMP.
3. Attack a system protected by the firewall, using a necessary hole in the firewall itself that grants must-have access to a box such as a web server.

The two attacks we choose are: 1 (attacking the firewall itself) and 2 (DoS via TCP SYN flood). We avoided 3 because we do not know the OS or versions of software on machines like the web server and mail server. We could use nmap to fingerprint the systems, and then research vulnerabilities from there. But we leave that as a fall-back option or perhaps something to do on a rainy day. Also, we relate that we have strongly considered attacking the VPN server, though it actually goes around the perimeter defense and not through it, and as such would not truly test the firewall. We hope primarily to demonstrate that we can penetrate not just the router, but the firewall itself.

We chose to attack the firewall because it gives us 2 primary points of failure in one box: the OS and the firewall software. It should be relatively simple to identify any exploits available for either. We chose a TCP SYN DoS because we know that there are certain paths open through the firewall which do not require us to use the evidently blocked ICMP protocol. An added bonus is that the SYN attack may yield more information about potential targets behind the firewall. But our primary hope is to make something go down. If it is a web server, DNS server, or some other server that goes down, wonderful. If we take down the border router or firewall, even better. If we make the firewall turn stupid and just allow everything to pass through (which is worse than being down), we will be elated.

Attack 1: The Firewall

Firewall-1, though known for its robustness and ease-of-use, has had a spotty record when it comes to security, despite the pretty GUI (thanks to Chris Brenton for that quote-to-remember). We have found these potential vulnerabilities in Firewall-1.

A. Format Strings vulnerability, found at
http://www.checkpoint.com/techsupport/alerts/format_strings.html
From the technical notes:

> A security issue exists in VPN-1/FireWall-1 version 4.1 whereby a valid firewall administrator connecting from an authorized management client may send malicious data to a management station inside a control connection, possibly preventing proper operation of the management station. This issue exists because some instances of improper string formatting occur in VPN-1/FireWall-1 version 4.1. By sending specially constructed commands through authorized communication channels, arbitrary code may be inserted onto the operating system stack of a VPN-1/FireWall-1 management station.[iii]

This attack could only come from a valid connection by authorized administrator. So, unless we are able to do some social engineering to gain such access, we consider this hole a challenge for some other day. Nonetheless, a disgruntled or otherwise malicious administrator has this hack at his disposal. Gruntled administrators should beware.

This hole is fixed by service pack 5. At the time of her practical, Donna was on service pack 3. Since she passed and is now certified, we assume she would patch Firewall-1 to keep the box safe.

B. RDP Communication vulnerability, found at
http://www.checkpoint.com/techsupport/alerts/rdp.html

Its description is:

> Check Point uses a proprietary protocol called RDP (UDP/259) for some internal communication between software components (this is not the same RDP as IP protocol 27). By default, VPN-1/FireWall-1 allows RDP packets to traverse firewall gateways in order to simplify encryption setup. Under some conditions, packets with RDP headers could be constructed which would be allowed across a VPN-1/FireWall-1 gateway without being explicitly allowed by the rule base.[iv]

Our first thought is, "Oops." Hmm. A carefully-crafted attack using some form of packet-mangling could get us past the firewall undetected and, once we have beachhead, allow us to set up shop for more malice.

This attack is addressed by service pack 4, and therefore inclusively in service pack 5. If Donna earns her raises, we might be out of luck with this one, too. But it sounds like a good starting place.

C. Another RDP issue, found at
http://www.checkpoint.com/techsupport/alerts/rdp_comms.html
The attack is described thus:

> Check Point has become aware of a condition with RDP Protocol in VPN-1/ FireWall-1 4.1 and Next Generation (NG) that may affect system stability. If the error occurs on a 4.1 module, certain management functions, such as logging and administrator communications, will halt. On NG modules, encryption key processing may be briefly interrupted.[v]

On the surface, this problem does not seem to be much of a goody for us. But consider: if logging and admin communications stop, we could couple this attack with some network mapping or other probing and scanning that might otherwise get logged by the firewall. The tech note states that the firewall rules are still enforced, so what we would hope for is an opportunity to discover silently some device we can exploit. Doing such a scan at other times might get us logged, and that's bad for us.

D. Finally, we look at one fairly recent Solaris exploit which targets SNMP, found at
http://www.cert.org/advisories/CA-2001-24.html
CERT's description is:

> *ovactiond* is the SNMP trap and event handler for both OpenView and NetView. There is a vulnerability in ovactiond that allows an intruder to execute arbitrary commands by sending a malicious message to the management server. These commands run with the privileges of the ovactiond process, which varies according to the operating system.[vi]

We note that Donna us utilizing SNMP on her network. Perhaps she runs OpenView or NetView on the firewall for stats. It's not likely, but it's possible. If a few things fall in place, this hole could allow us to disable Firewall-1 altogether. Once the strong man is bound, it's easier to rob the house.

All things considered, we choose exploit B, the RDP hole in Firewall-1. It is as likely to be available to us as the others. Perhaps Donna moved on to bigger and better things and the shop isn't kept up quite as well as it was in her heyday. To start, we research packet manipulation tools. The following three resources were invaluable:

http://www.packetforge.net/packet/index.html
http://www.packetfactory.net/
http://www.nessus.org/.

On packetforge, we found Spak to be a rather interesting creature. It can be downloaded from http://www.xenos.net/software/spak/[vii]. If it works as advertised, we can craft a packet to get us past Firewall-1's ruleset and hopefully find some eager recipient of a rootkit inside her firewall. It won't be easy, but it certainly could prove fruitful.

Our hope is to find either a hole in Solaris (on the firewall itself) that is available once we disable or bypass Firewall-1's protection, or a hole in some server protected by the firewall, a hole we can exploit in the absence of the firewall's protection. Although not our chosen method of attack, the aforementioned SNMP hole may be approachable from more than one angle. Communication with the SNMP daemon from the localhost to the localhost may be one path to get into the box. Or, if any of her servers allow services like FTP access that were previously blocked from the outside world by the firewall, we may now have a means of finding a hole in one of them, if we can bypass the firewall. Of course, this RDP hole we hope to exploit seems mostly to allow, in Checkpoint's words, simply "a surreptitious communication channel."[viii] But we won't know until we try.

Again, patching the firewall would (likely) stop us dead. Donna and her staff could also drop port 259/UDP at the border router if that wouldn't cause administrative difficulties for them. To be versatile, perhaps we could start out with attack C, mapping out the network when logging is down, then implement this attack, possibly yielding us some path in.

Attack 2: DoS via TCP SYN flood

A SYN flood could afford us a few more opportunities to access Donna's network, or, at the least, take some elements of it out of commission for periods of time. To get us on our way, we reviewed an older document found at http://www.fc.net/phrack/files/p48/p48-13.html. It promotes itself as "a comprehensive analysis of TCP SYN flooding"[ix] and includes details and source code for carrying out the attack. Other writings on the subject covered basically the same ground.

To start, we must either find or create some tool to do all this dirty work for us. Assuming we have 50 compromised systems behind cable modem or DSL connections, and that we have the tool that creates the SYN flood pre-installed on these systems, we must simply choose a good time to start the attack. We can choose a heavy traffic time, which may make our job easier. Or we can choose a time when there is likely nobody around that has access to the systems and could shut us down.

We also have several avenues into the network. We think it best to attack a machine with several TCP services running which are publicly accessible, or at least running services which receive a lot of traffic. At the least, we will attack the web server. We may even find the router, the firewall, or the VPN server (a trusted machine, too) vulnerable. In any case, a successful attack will cause Donna's network downtime, and may let us in some back door.

Most OS releases currently have patches to stop this sort of attack. Moreover, stateful packet filtering pretty much takes the teeth out of this type of attack. Nonetheless, since we intend to attack services which are allowed by the firewall, we may be able to carry out this attack undetected for quite some time.

In the event that these machines are patched current, we will be disappointed in our venture but pleased with the dedication Donna and her staff have to protecting their network.

Appendix A - .config file from firewall setup

Portions of this config file not pertinent to networking/firewalling have been excluded.

```
#
# Networking options
#
CONFIG_PACKET=y
CONFIG_PACKET_MMAP=y
CONFIG_NETLINK=y
CONFIG_RTNETLINK=y
CONFIG_NETLINK_DEV=y
CONFIG_NETFILTER=y
# CONFIG_NETFILTER_DEBUG is not set
CONFIG_FILTER=y
CONFIG_UNIX=y
CONFIG_INET=y
CONFIG_TUX=m
CONFIG_TUX_EXTCGI=y
# CONFIG_TUX_EXTENDED_LOG is not set
# CONFIG_TUX_DEBUG is not set
CONFIG_IP_MULTICAST=y
CONFIG_IP_ADVANCED_ROUTER=y
CONFIG_RTNETLINK=y
CONFIG_NETLINK=y
CONFIG_IP_MULTIPLE_TABLES=y
CONFIG_IP_ROUTE_FWMARK=y
CONFIG_IP_ROUTE_NAT=y
CONFIG_IP_ROUTE_MULTIPATH=y
CONFIG_IP_ROUTE_TOS=y
CONFIG_IP_ROUTE_VERBOSE=y
CONFIG_IP_ROUTE_LARGE_TABLES=y
# CONFIG_IP_PNP is not set
CONFIG_NET_IPIP=m
CONFIG_NET_IPGRE=m
CONFIG_NET_IPGRE_BROADCAST=y
CONFIG_IP_MROUTE=y
CONFIG_IP_PIMSM_V1=y
CONFIG_IP_PIMSM_V2=y
# CONFIG_ARPD is not set
CONFIG_INET_ECN=y
CONFIG_SYN_COOKIES=y

#
#   IP: Netfilter Configuration
#
CONFIG_IP_NF_CONNTRACK=m
CONFIG_IP_NF_FTP=m
CONFIG_IP_NF_IRC=m
CONFIG_IP_NF_QUEUE=m
CONFIG_IP_NF_IPTABLES=m
CONFIG_IP_NF_MATCH_LIMIT=m
CONFIG_IP_NF_MATCH_MAC=m
CONFIG_IP_NF_MATCH_MARK=m
```

```
CONFIG_IP_NF_MATCH_MULTIPORT=m
CONFIG_IP_NF_MATCH_TOS=m
CONFIG_IP_NF_MATCH_TCPMSS=m
CONFIG_IP_NF_MATCH_STATE=m
CONFIG_IP_NF_MATCH_UNCLEAN=m
CONFIG_IP_NF_MATCH_OWNER=m
CONFIG_IP_NF_FILTER=m
CONFIG_IP_NF_TARGET_REJECT=m
CONFIG_IP_NF_TARGET_MIRROR=m
CONFIG_IP_NF_NAT=m
CONFIG_IP_NF_NAT_NEEDED=y
CONFIG_IP_NF_TARGET_MASQUERADE=m
CONFIG_IP_NF_TARGET_REDIRECT=m
CONFIG_IP_NF_NAT_IRC=m
CONFIG_IP_NF_NAT_FTP=m
CONFIG_IP_NF_MANGLE=m
CONFIG_IP_NF_TARGET_TOS=m
CONFIG_IP_NF_TARGET_MARK=m
CONFIG_IP_NF_TARGET_LOG=m
CONFIG_IP_NF_TARGET_TCPMSS=m
CONFIG_IP_NF_COMPAT_IPCHAINS=m
CONFIG_IP_NF_NAT_NEEDED=y
CONFIG_IP_NF_COMPAT_IPFWADM=m
CONFIG_IP_NF_NAT_NEEDED=y

#
#   IP: Virtual Server Configuration
#
CONFIG_IP_VS=m
# CONFIG_IP_VS_DEBUG is not set
CONFIG_IP_VS_TAB_BITS=16
CONFIG_IP_VS_RR=m
CONFIG_IP_VS_WRR=m
CONFIG_IP_VS_LC=m
CONFIG_IP_VS_WLC=m
CONFIG_IP_VS_LBLC=m
CONFIG_IP_VS_LBLCR=m
CONFIG_IP_VS_DH=m
CONFIG_IP_VS_SH=m
CONFIG_IP_VS_FTP=m
# CONFIG_IPV6 is not set
# CONFIG_KHTTPD is not set
# CONFIG_ATM is not set
# CONFIG_IPX is not set
# CONFIG_ATALK is not set
# CONFIG_DECNET is not set
# CONFIG_BRIDGE is not set
# CONFIG_X25 is not set
# CONFIG_LAPB is not set
# CONFIG_LLC is not set
# CONFIG_NET_DIVERT is not set
# CONFIG_ECONET is not set
# CONFIG_WAN_ROUTER is not set
# CONFIG_NET_FASTROUTE is not set
# CONFIG_NET_HW_FLOWCONTROL is not set
```

```
#
# QoS and/or fair queueing
#
CONFIG_NET_SCHED=y
CONFIG_NETLINK=y
CONFIG_RTNETLINK=y
CONFIG_NET_SCH_CBQ=m
CONFIG_NET_SCH_CSZ=m
CONFIG_NET_SCH_PRIO=m
CONFIG_NET_SCH_RED=m
CONFIG_NET_SCH_SFQ=m
CONFIG_NET_SCH_TEQL=m
CONFIG_NET_SCH_TBF=m
CONFIG_NET_SCH_GRED=m
CONFIG_NET_SCH_DSMARK=m
CONFIG_NET_SCH_INGRESS=m
CONFIG_NET_QOS=y
CONFIG_NET_ESTIMATOR=y
CONFIG_NET_CLS=y
CONFIG_NET_CLS_TCINDEX=m
CONFIG_NET_CLS_ROUTE4=m
CONFIG_NET_CLS_ROUTE=y
CONFIG_NET_CLS_FW=m
CONFIG_NET_CLS_U32=m
CONFIG_NET_CLS_RSVP=m
CONFIG_NET_CLS_RSVP6=m
CONFIG_NET_CLS_POLICE=y

#
# Network device support
#
CONFIG_NETDEVICES=y

#
# ARCnet devices
#
# CONFIG_ARCNET is not set
CONFIG_DUMMY=m
CONFIG_BONDING=m
CONFIG_EQUALIZER=m
CONFIG_TUN=m
CONFIG_ETHERTAP=m
CONFIG_NET_SB1000=m

#
# Ethernet (10 or 100Mbit)
#
CONFIG_NET_ETHERNET=y
# CONFIG_SUNLANCE is not set
CONFIG_HAPPYMEAL=m
# CONFIG_SUNBMAC is not set
# CONFIG_SUNQE is not set
# CONFIG_SUNLANCE is not set
CONFIG_SUNGEM=m
CONFIG_NET_VENDOR_3COM=y
CONFIG_EL1=m
```

```
CONFIG_EL2=m
CONFIG_ELPLUS=m
CONFIG_EL16=m
CONFIG_EL3=m
CONFIG_3C515=m
# CONFIG_ELMC is not set
# CONFIG_ELMC_II is not set
CONFIG_VORTEX=m
CONFIG_LANCE=m
CONFIG_NET_VENDOR_SMC=y
CONFIG_WD80x3=m
# CONFIG_ULTRAMCA is not set
CONFIG_ULTRA=m
# CONFIG_ULTRA32 is not set
# CONFIG_SMC9194 is not set
CONFIG_NET_VENDOR_RACAL=y
CONFIG_NI5010=m
CONFIG_NI52=m
CONFIG_NI65=m
CONFIG_AT1700=m
CONFIG_DEPCA=m
CONFIG_HP100=m
CONFIG_NET_ISA=y
CONFIG_E2100=m
CONFIG_EWRK3=m
CONFIG_EEXPRESS=m
CONFIG_EEXPRESS_PRO=m
CONFIG_HPLAN_PLUS=m
CONFIG_HPLAN=m
CONFIG_LP486E=m
CONFIG_ETH16I=m
CONFIG_NE2000=m
CONFIG_NET_PCI=y
CONFIG_PCNET32=m
CONFIG_ADAPTEC_STARFIRE=m
CONFIG_AC3200=m
CONFIG_APRICOT=m
CONFIG_CS89x0=m
CONFIG_TULIP=m
# CONFIG_TULIP_MWI is not set
CONFIG_TULIP_MMIO=m
CONFIG_DE4X5=m
CONFIG_DGRS=m
CONFIG_DM9102=m
CONFIG_EEPRO100=m
# CONFIG_LNE390 is not set
# CONFIG_FEALNX is not set
CONFIG_NATSEMI=m
CONFIG_NE2K_PCI=m
# CONFIG_NE3210 is not set
# CONFIG_ES3210 is not set
CONFIG_8139TOO=m
# CONFIG_8139TOO_PIO is not set
# CONFIG_8139TOO_TUNE_TWISTER is not set
CONFIG_8139TOO_8129=y
CONFIG_SIS900=m
```

```
CONFIG_EPIC100=m
CONFIG_SUNDANCE=m
CONFIG_TLAN=m
CONFIG_VIA_RHINE=m
CONFIG_WINBOND_840=m
# CONFIG_LAN_SAA9730 is not set
CONFIG_NET_POCKET=y
CONFIG_ATP=m
CONFIG_DE600=m
CONFIG_DE620=m

#
# Ethernet (1000 Mbit)
#
CONFIG_ACENIC=m
# CONFIG_ACENIC_OMIT_TIGON_I is not set
CONFIG_DL2K=m
# CONFIG_MYRI_SBUS is not set
CONFIG_NS83820=m
CONFIG_HAMACHI=m
CONFIG_YELLOWFIN=m
CONFIG_SK98LIN=m
# CONFIG_FDDI is not set
# CONFIG_HIPPI is not set
# CONFIG_PLIP is not set
# CONFIG_PPP is not set
# CONFIG_SLIP is not set
```

References

[i] Linux router project home page, URL: http://master-www.linuxrouter.org:8080/ (11/29/2001)

[ii] Jackson, Josh. "Coyote Linux Products."
URL: http://www.coyotelinux.com/products.php (11/29/2001)

[iii] "Alerts: Format Strings." URL: http://www.checkpoint.com/techsupport/alerts/format_strings.html (12/26/2001)

[iv] "Alerts: RDP Communication." URL: http://www.checkpoint.com/techsupport/alerts/rdp.html (12/26/2001)

[v] "RDP Communications Issue." URL: http://www.checkpoint.com/techsupport/alerts/rdp_comms.html (12/26/2001)

[vi] "CERT® Advisory CA-2001-24 Vulnerability in OpenView and NetView" URL: http://www.cert.org/advisories/CA-2001-24.html (12/26/2001)

[vii] "SPAK." URL: http://www.xenos.net/software/spak/ (12/26/2001)

[viii] "Alerts: RDP Communication." URL: http://www.checkpoint.com/techsupport/alerts/rdp.html (12/26/2001)

[ix] "Project Neptune." Phrack Magazine. July 1996. URL: http://www.fc.net/phrack/files/p48/p48-13.html (12/26/2001)