



# Global Information Assurance Certification Paper

Copyright SANS Institute  
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

Table of Contents .....	1
Vince_Streiff_GCFW.doc.....	2

© SANS Institute 2000 - 2002, Author retains full rights.

***GIAC Certified Firewall Analyst (GCFW) Practical Assignment  
v.1.7***

# **GIAC Enterprises**

**Security for a Small Company**

**By Vincent R. Streiff**

**Fall 2002**

© SANS Institute 2000 - 2002, Author retains full rights.

(this page left intentionally blank)

## Table of Contents

<u>OVERVIEW</u>	2
<u>Assignment 1: Security Architecture</u>	3
<u>Summary of the problem</u>	3
<u>Analysis and Goals</u>	6
<u>1. Customers</u>	6
<u>2. Suppliers</u>	7
<u>3. Partners</u>	7
<u>4. GIAC Enterprises' On-site Employees</u>	8
<u>5. GIAC Enterprises Mobile Safes Force &amp; Off-site Employees</u>	9
<u>6. Potential clients and other on-site visitors</u>	10
<u>New Network Architecture</u>	12
<u>Summary of new security policy</u>	14
<u>Assignment 2: Security Policy &amp; Tutorial</u>	18
<u>Overview</u>	18
<u>Access requirements by component:</u>	18
<u>1. The Border Router:</u>	20
<u>A Tutorial for Configuring the Netopia R5300</u>	20
<u>2. The Primary Firewall</u>	38
<u>A. Firewall A:</u>	39
<u>B. Firewall B:</u>	73
<u>3. VPN(s)</u>	84
<u>A. Windows 2000 Routing &amp; Remote Access (RRAS) using L2TP over IPSec.</u>	85
<u>Real World implementation hints:</u>	116
<u>B. Citrix Nfuse v. 1.7</u>	117
<u>C. SSH-2</u>	119
<u>Assignment 3: Verify the Firewall Policy</u>	120
<u>Overview</u>	120
<u>1. The Plan</u>	120
<u>Audit costs and risks:</u>	123
<u>2. Conducting the Audit</u>	123
<u>3. Evaluating the Audit</u>	141
<u>Assignment 4: Design under fire</u>	149
<u>Overview</u>	150
<u>A. Attacking the firewall</u>	150
<u>Result:</u>	152
<u>B. Denial of Service Attack</u>	152
<u>Result:</u>	154
<u>Countermeasure:</u>	154
<u>C. Compromise a machine through Perimeter</u>	154
<u>Result:</u>	156
<u>Conclusions:</u>	157

<u>Bibliography</u>	158
<u>Books &amp; other print materials</u>	158
<u>On-line references</u>	158

© SANS Institute 2000 - 2002, Author retains full rights.

## OVERVIEW

This paper is intended to show how even a fairly small company can have excellent security.

The GCFW Practical Assignment version 1.7 has 4 parts:

### **Assignment 1 – Security Architecture (15 points)**

Define a network security architecture for GIAC Enterprises, an e-business which deals in the online sale of fortune cookie sayings.

### **Assignment 2 – Security Policy and Tutorial (35 points)**

Based on the security architecture that you defined in Assignment 1, provide a security policy for the following three components:

- Border Router(s)
- Primary Firewall(s)
- VPN(s)

### **Assignment 3 – Verify the Firewall Policy (25 points)**

Conduct a technical audit of GIAC's primary firewall in order to verify that the policies are correctly enforced as described in Assignments 1 and 2.

### **Assignment 4 – Design Under Fire (25 points)**

Select a network design from any [GCFW practical](#) posted in the previous 6 months; research and design the following three types of attacks against the architecture:

1. An attack against the firewall itself.
2. A denial of service attack.
3. An attack plan to compromise an internal system through the perimeter system.



# Assignment 1: Security Architecture

## *Summary of the problem*

GIAC Enterprises has recently hired a new IT Manager, who immediately recognized the need for significantly improved security. However, the very slim profit margins of the fortune cookie business, combined with the current sagging economy, require these changes to be made at minimum expense. For this reason, the newly recognized needs will be met, wherever possible, using existing equipment and software. With very few exceptions, open source software will be used for any software needs not met by the current inventory, in order to avoid unnecessary expenses. Red Hat Linux 7.3<sup>1</sup> (RH) will be used as the open source platform for these purposes; while there are other open source operating systems and other versions of Linux designed with security in mind, Red Hat is the version the IT Manager is familiar with. Because each additional operating system adds complexity, GIAC Enterprises will standardize on two operating systems: Red Hat Linux 7.3 where feasible or prudent, Microsoft Windows 2000 elsewhere.

The pre-existing infrastructure for GIAC Enterprises was fairly simple. The main office, which includes the corporate offices, holds a staff of 37 (though this fluctuates somewhat as temporary help is occasionally hired for data entry.) The previous IT Manager – self-taught on the job – managed to keep things running remarkably well, but security was not one of his priorities.

The corporate offices are running Windows 2000 entirely, with a single Active Directory forest with a single domain. Messaging is handled by Exchange 2000. There was a firewall (Elron Software's Firewall 3.04<sup>2</sup>), running on a Windows NT machine, but no DMZ of any sort. This firewall has some nice features and appeared to be working, but its logging capabilities were severely lacking, and there was no intrusion detection system to confirm how well the firewall really was, or was not, performing. It was also crashing rather frequently, disrupting the office's access to the Internet. Replacing this firewall was a top priority.

The office is connected to the Internet via a T-1 connection; the router is a Netopia R5300. This router has limited packet filtering abilities, but those abilities had never been implemented. This router was also monitored and maintained via telnet up to this point, as the R5300 lacked the capability of any form of secure remote management.<sup>3</sup> Due to the severe security threats sending the border router's access information in plain text creates, this practice was immediately stopped, and the router's remote access capabilities were

<sup>1</sup> <http://www.redhat.com/>

<sup>2</sup> <http://www.elronsoftware.com/productfamily/firewall.shtml>

<sup>3</sup> PPTP capability was added with firmware version 4.4; IPSec support was added with version 4.8 -- see [http://www.netopia.com/en-us/equipment/tech/fmw\\_features.html](http://www.netopia.com/en-us/equipment/tech/fmw_features.html)

turned off until after the firmware was upgraded and the telnet sessions could be secured using PPTP. All router maintenance until then must be done via a terminal directly connected to the router via serial cable. While inconvenient, the security risks of using telnet were unacceptable. The IT Manager wanted to replace the router immediately with more capable Cisco equipment; his request was denied, however. A new router has instead been budgeted for next year; in the meantime, the IT Manager will implement as much packet filtering on the Netopia R5300 as performance will allow.

Citrix Metaframe had been recently implemented for both remote access and internal use; however, the majority of clients were still using the old Windows 2000 RRAS (PPTP) server for remote access, because it enables them to compose e-mail off-line, and then synchronize MS Outlook when they have an Internet connection available. The IT Manager was able to significantly improve both security and functionality of the remote access sessions by reconfiguring the existing equipment; rather than PPTP, L2TP over IPsec will now be used.

The single biggest security concern, however, was the lack of physical security. GIAC Enterprises' servers and other critical networking equipment were located in the large bay area that served as the previous IT Manager's office. This carpeted, open area served as a connector between two other hallways – in effect, the servers themselves were located in a hallway anyone with access to the floor had access to. This included the cleaning crew, who of course needed to vacuum the carpet the server racks were sitting on. After recovering from the shock, the new IT Manager explained the futility of trying to secure the computer systems through software alone. Once they understood that anyone with physical access could do pretty much anything they wanted, upper management approved immediate restructuring of the computer facilities. Space vacated during "rightsizing" was used to build a dedicated, secured (and un-carpeted) network operations center. Building this secure area was the one large expense approved in this endeavor.

We can see a sketch of the inherited network in *Figure 1*.

© SANS Institute

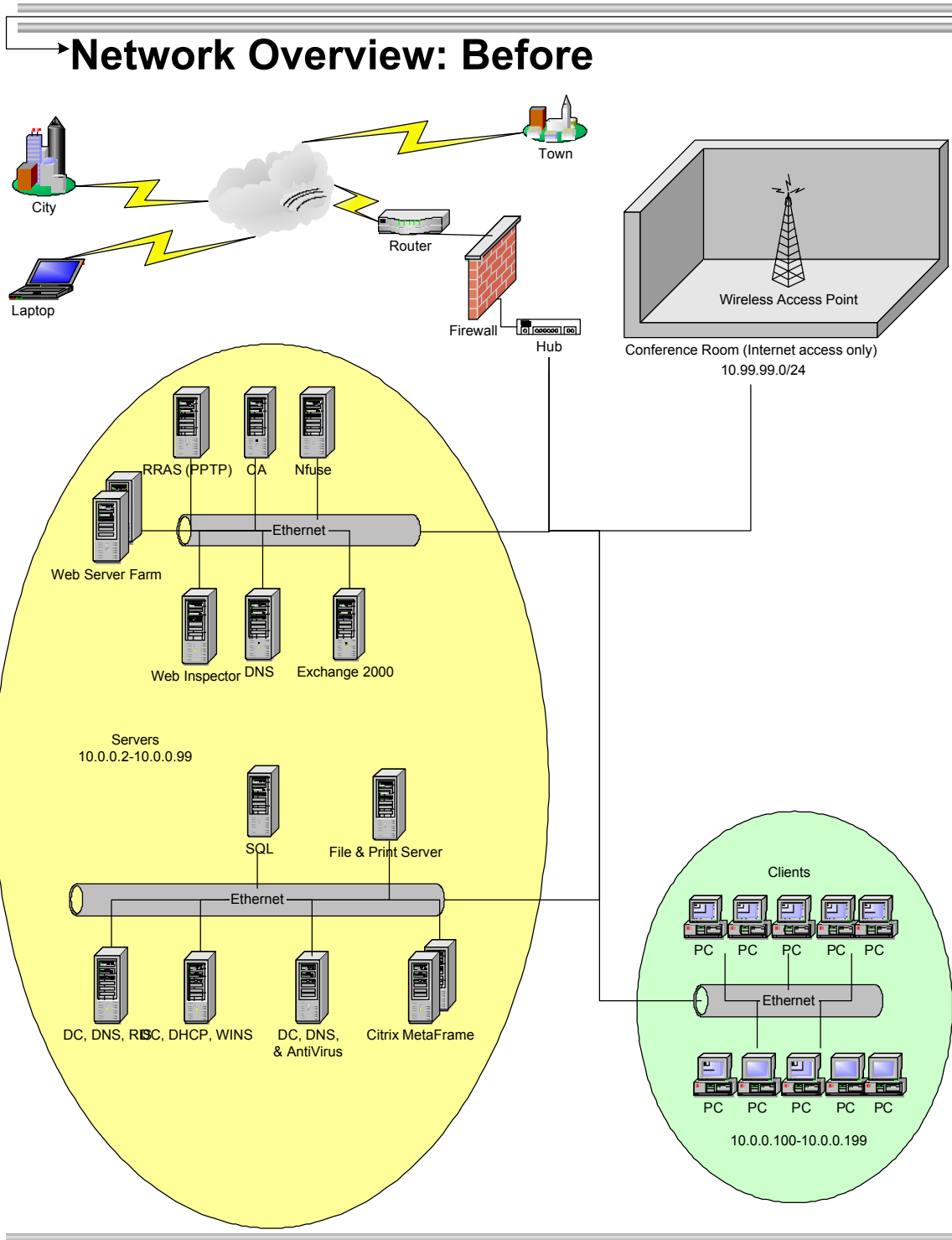


Figure 1

## ***Analysis and Goals***

Before making any changes, the new IT Manager met with all of the corporate officers to determine GIAC Enterprises' core business needs. These will guide both the architecture and the implementation of the new security plan. During this needs analysis, GIAC Enterprises identified the following classifications of users of their computer facilities and services:

1. *Customers*, who require access to the GIAC Enterprises Web site, and who send e-mail to GIAC Enterprises staff.
2. *Suppliers*, particularly those who submit the fortune cookie sayings. GIAC Enterprises' suppliers of other materials – paper, dough ingredients, etc. – do not require direct or special access to the network; inventory is recorded by staff manually. (There has been some discussion of automating this process in the future, however, so the new architecture will require flexibility to incorporate this later on without requiring any architectural changes.)
3. *Partners*, which are international companies that translate and resell the fortunes.
4. GIAC Enterprises' *on-site employees*.
5. GIAC Enterprises' *mobile sales force* and other *off-site employees*.
6. *Potential clients* and other on-site visitors will be provided access to the Internet via connections in the conference room.

The new IT Manager then proceeded to define the following security architecture and general policy, based on the needs of each of the 6 groups of users. The new policy begins with an assumption that any access not specifically required will be denied. In addition, naturally, there is an assumption that GIAC Enterprises needs to continue operations during this transition. Lastly, the IT Manager was instructed to spend an absolute minimum amount of money in this endeavor.

### **1. Customers**

GIAC Enterprises' customers only need access to the public Web site, which enables online orders via SSL. All data from the transactions is stored in a back-end SQL database. The Web server (Windows 2000 Server running IIS) and the database server (Microsoft SQL Server 2000) will use IPsec to secure their communications with each other. These servers will be "hardened" to the greatest extent possible, and the logs will be closely monitored.

The only connectivity required through the external firewall for the customer traffic is TCP ports 80 (HTTP) and 443 (HTTPS). The Web server and the SQL server housing the data will be located on a service network, or DMZ. Traffic from the Web server that is neither a reply to an existing Internet connection nor

destined to the SQL server will generate immediate alerts in the new intrusion detection system.

## **2. Suppliers**

Suppliers have traditionally submitted their sayings by simply e-mailing MS Word documents to GIAC Enterprises. Because of the constant, pervasive threat of Word macro viruses & worms, the security concerns with sending core business information across the Internet in e-mail, and because this process makes tracking submissions very difficult, a new system will be put in place to enable suppliers to submit their sayings online via a standard form. The existing Web and SQL servers will be leveraged to provide this service, with a goal of moving them to separate, dedicated hardware as soon as funding allows. Suppliers will upload files in comma-separated or tab-delimited format; they will then be able to review and edit them online, before confirming the submission. In addition, GIAC Enterprises' staffs' computers will have MS Outlook 2002 re-configured to its default setting of not allowing access to .DOC attachments. (The previous IT Manager had enabled access to attachments with MS Office extensions for convenience to staff.)

This submitted data will be stored on the Microsoft SQL Server 2000 backend database. [While there is definitely concern about having so much of GIAC Enterprises' critical business information in the DMZ, the server was put there because of the assumption that in the event of a server compromise, the damage would hopefully be limited to the DMZ. The traffic from the Web server to the SQL server is the same regardless of the SQL server's placement; if the SQL server were in the internal network, and it were to be successfully attacked, a compromised server would then be inside the internal office network.] Storing the data in the SQL server will facilitate vastly improved tracking of individual supplier submissions, for purposes of tracking finances, quality assurance, etc. (Though it's never discussed outside of the office, there is some concern among GIAC Enterprises' accountants that the current system of e-mail and Word documents lends itself to fraud, because the accounting department doesn't have the ability to easily cross-check the suppliers' invoices with the goods actually delivered.) The IT Manager expressed his concern over storing the company's key proprietary info – the fortune sayings – on a server located in the DMZ, but was told not to change too much at once.

## **3. Partners**

The international companies that translate and resell GIAC Enterprises' fortunes have been getting those fortunes e-mailed to them in Word format – often, the Word documents submitted by GIAC Enterprises' suppliers were simply forwarded to the partners. In addition to the security concerns with sending their core business information across the globe in e-mail, there has been at least one embarrassing instance of a staff member forwarding a Word document that

contained the supplier's private information. Partners will now, therefore, download the sayings via the Web site, from the same database the suppliers submit them to. (Only records cleared by GIAC Enterprises staff will be available for them to download, however.) This will, again, significantly improve both security and tracking of accounts payable and accounts receivable. It's also hoped these improvements in accounting will improve the general demeanor of the accounting department, though the staff views this as a long shot.

Partners do not need access to any other information. There was hope for generating additional revenue by licensing GIAC Enterprises' unique dough formula, but that has been put on hold due to the varying local food regulations of the partners' host countries.

#### **4. GIAC Enterprises' On-site Employees**

Employees on-site obviously have the most extensive access requirements to the network. Their access, however, does not need to be unlimited. Access to specific resources will be based on job function and department; individuals will not be given access to any resources they don't need to perform their job. One obvious example is that only members of the accounting department will have access to the financial records.

General usage policies are outside the scope of this document. In a nutshell, limited personal use is allowed, as long as it does not interfere with work; no software of any kind may be installed by non-IT staff, however, and the uses of instant messaging, chat, peer to peer and personal Web mail are strictly prohibited.

Due to previous abuses, Internet usage is monitored via Elron Software's Web Inspector, currently at version 6.01. Accessing sites that contain questionable material, such as pornography or other adult content, sends an alert via e-mail to the HR department. Sites are not blocked, due to the concern for false positives – we don't want to accidentally block CNN, for example. E-mail traffic is not currently monitored, though there has been some discussion of implementing either Elron Software's Message Inspector (<http://www.elronsoftware.com/productfamily/msginspector.shtml>) or SurfControl's E-mail Filter software (<http://www.surfcontrol.com/products/email/>).

In order to assist with stripping out any Java, ActiveX, or other potentially harmful code, all out-bound Web traffic will be routed through a proxy. The firewall will therefore be configured to only allow outbound Web traffic from the internal network to go to the service network. (In fact, all outbound traffic of any sort from the internal network to the Internet must be routed through the DMZ.) It was determined that there is no legitimate business reason for the majority of

staff to have the ability to download files from the Internet; therefore, Passive FTP access will be available only from designated machines which only IT staff have permission to log onto. Due to their determination to monitor regulatory affairs via the Internet, people in the Advocacy Department have requested the ability to use RealAudio and Windows Media Player for monitoring events on Capitol Hill. (The IT Manager's suggestions that the existing cable TV's be used instead were called tactless, and ignored.) Both of these services will therefore be allowed, but configured to operate over HTTP, so no additional holes in the firewall will be needed. The client software – Windows Media Player in particular – will be closely monitored to ensure any and all security flaws are patched or otherwise addressed immediately.

GIAC Enterprises also hires temporary staff occasionally for data entry. Temps use well-hardened Windows 2000 Professional workstations to work from (they meet or exceed the CIS Gold Standard.<sup>4</sup>) In addition, Active Directory's Group Policy is used to restrict them to running only the programs they need. These workers are on-site only; they are denied Remote Access capability.

Internal staff who need to work on or with the SQL server will use IPSec to communicate with it. This not only secures the communications, but also simplifies the firewall's rulesets; we only need to allow UDP 500 for IKE, and protocols 50 (ESP) and 51 (AH) between the SQL server and the internal network. These communications may be initiated from the internal network only; the SQL server will not be able to initiate communication with the internal network.

## **5. GIAC Enterprises Mobile Safes Force & Off-site Employees**

The majority of remote access for staff will be provided via Citrix MetaFrame XPa, Feature Release 2<sup>5</sup>. This is the one large IT expenditure GIAC Enterprises had this year, because the cost of implementation was not much higher than the normal annual replacement of 1/3 of the computers. (GIAC Enterprises has been on a 3-year replacement cycle for its PC's for the past 4 years, and generally spent in the neighborhood of \$20,000 on PC replacements. By doing a significant amount of the work themselves, GIAC Enterprises implemented Citrix Metaframe at a cost of roughly \$30,000. Because Citrix MetaFrame enables the implementation of a "thin client" solution for the majority of staff, it will eliminate the need for annual PC replacements; GIAC Enterprises expects to see a return on investment of over \$10,000 next year alone.) GIAC Enterprises recognizes that Web-based services, such as a future .NET version of Microsoft Office that doesn't require local installation, may be cheaper than Citrix when they are available. Truly Web-based office productivity applications won't be available for implementation next year, however, and the security implications of such an architecture have not yet been adequately addressed;

<sup>4</sup> [http://esrc.nist.gov/itsec/guidance\\_W2Kpro.html](http://esrc.nist.gov/itsec/guidance_W2Kpro.html)

<sup>5</sup> <http://www.citrix.com/products/metaframexp.asp>

Citrix still makes sense.

It also improves remote access security significantly, as the majority of needs can be met through the Citrix NFuse Web interface, combined with Citrix Secure Gateway (CSG). The implementation the IT Manager inherited, however, did not take advantage of CSG, nor did the NFuse site use SSL. This resulted in staff sending usernames and passwords over the Internet in plain text, plus required opening additional holes in the firewall to enable ICA connections (tcp port 1494) directly to each of the Citrix MetaFrame servers. That situation was immediately rectified to prevent sending passwords in unencrypted clear text, and all users who had logged on were required to change their passwords. Staff is excited that use of Citrix for remote access also means that they will see the exact same settings for their programs whether they are in the office or working from home; IT is looking forward to utilizing the "session shadowing" capabilities to save on shoe leather during staff support calls.

Citrix is not a panacea, however, and more traditional VPN access will still be required for the foreseeable future for the mobile sales force to synchronize MS Outlook. Because they often need to work on planes and other areas without Internet connectivity, they will use both Outlook's "Send and Receive" function and Windows 2000's "Offline Files" functionality. Because synchronizing Outlook can not be done via Citrix, VPN access of some sort is required. Due to the need to avoid additional expenses, GIAC Enterprises will continue to use the existing Windows 2000 RRAS service for this. However, both the server and the clients will be reconfigured to improve security. Instead of PPTP sessions with encryption based on user passwords, clients will now use L2TP over IPsec.<sup>6</sup> This will require authentication based on digital certificates as well as a username and password. [The IT Manager suggested implementing a smart card solution, but management balked at the expense.] The list of servers that can be accessed through the VPN server will be restricted to domain controllers and the Exchange server, in order to provide Outlook synchronization services and as little else as possible. Lastly, the VPN server will be moved to a separate, dedicated segment off the firewall in order to leverage Netfilter's logging and filtering capabilities.

IT Staff will also need remote access to the new Linux boxes for administrative purposes. This will be done via OpenSSH; because of the security flaws in SSH-1, only SSH-2 will be implemented. ([www.openssh.org](http://www.openssh.org))

## **6. Potential clients and other on-site visitors**

Connectivity to the Internet is provided for authorized guests in the conference room; an 802.11b wireless access point was installed there last fall. The Board of Directors has been advised this is a security risk, but they decided the need to provide the service outweighs the risks. As a compromise, the wireless access

<sup>6</sup> See RFC 2661, "Layer 2 Tunneling Protocol," at <http://www.ietf.org/rfc/rfc2661.txt>.



point will remain turned off when not in use, powered on only as needed. In addition, this service is being moved to a separate, isolated network, so that any unauthorized use of the wireless connection – wandering public, other tenants in the building, etc., who succeed in connecting to the wireless access point – will only succeed in getting Internet access. This will not only be on a separate network, but will indeed have a separate firewall, though it will be connected to the same router. The firewall will only allow outbound HTTP, HTTPS, and DNS traffic from this point. The logs of both this firewall and the wireless access point will be monitored for any questionable activity. [Because they will be used very infrequently, the logs will simply be checked manually. Though inconvenient, this was preferred over opening additional holes in the primary firewall to allow syslog.]

Due to the current impossibility of adequately securing wireless traffic, GIAC Enterprises' staff is prohibited from using this wireless access point for any work-related activity. If they must access the Internet while in the conference room, they are required to use one of the multiple wired Ethernet drops. If GIAC Enterprises' own staff needs access to anything on the internal network while in the conference room, they will simply connect via the encrypted Citrix NFuse remote access solution as if they were anywhere else outside of the internal network.

© SANS Institute 2000 - 2002,

## New Network Architecture

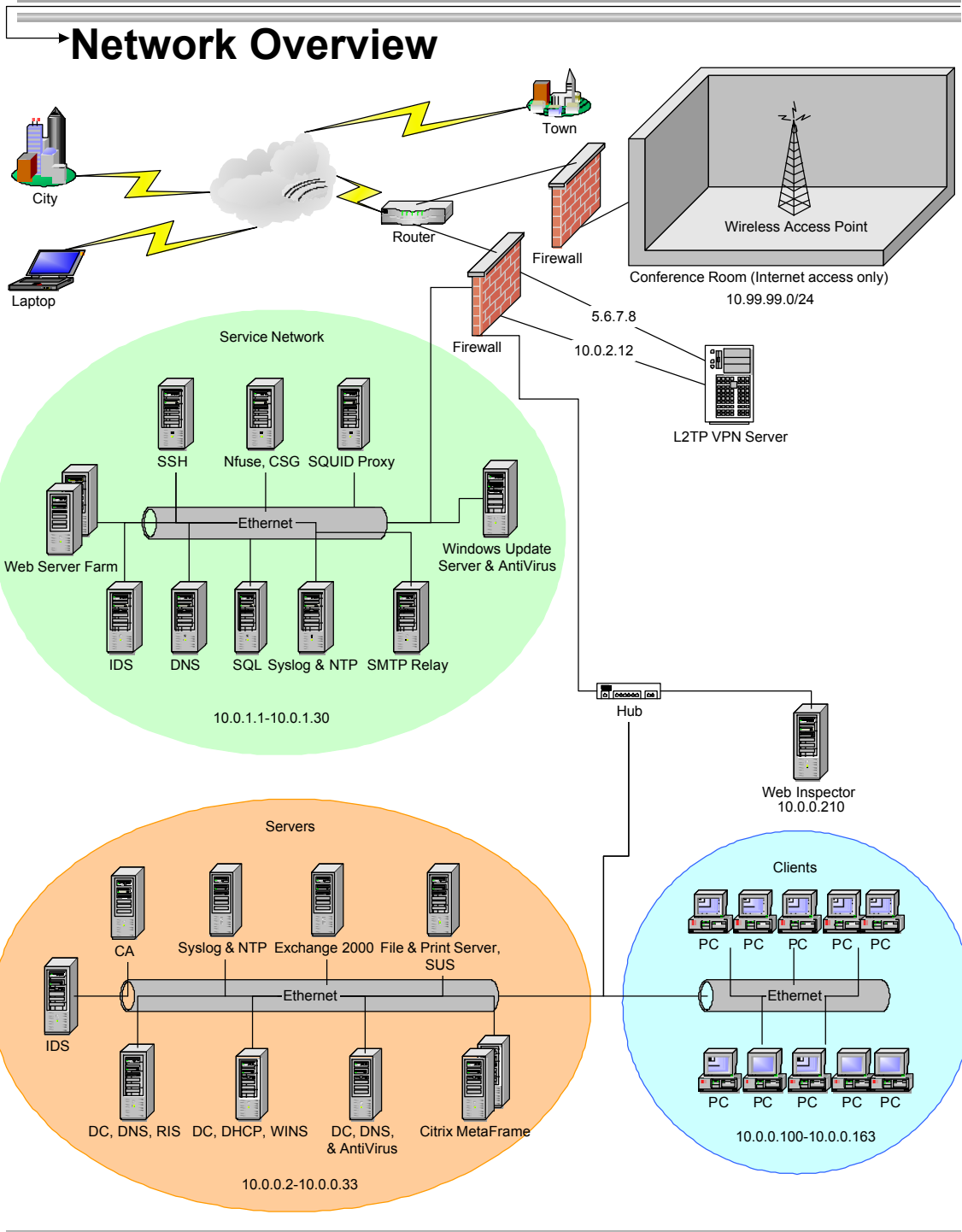


Figure 2

Several architectural changes are readily apparent in *Figure 2*. The most obvious is the separation of the one single network into 4 separate networks: one for the services accessible from the Internet, one for the internal office, one for the L2TP VPN server, and one for the conference room. (Note that not all client PC's are shown in this drawing.)

The router, a Netopia R5300, has not been replaced or moved. As it is still our connection to the Internet, making all the rest of our work possible (and necessary), it must of course be on the outer edge of our network. It is also a single point of failure; if our router fails, or is compromised, we have no other access to the Internet. We won't be able to get out, and our customers, partners, suppliers, and off-site staff won't be able to get to us. We will alter this equipment somewhat, however, by implementing packet filtering; this is described in much more detail later.

The old Elron Firewall on Windows NT has been replaced with Netfilter & iptables running on a Red Hat Linux 7.3 bastion host. This is directly behind the router, so that all traffic must go through the firewall immediately before or after traversing the Internet. This runs on the same hardware as the previous firewall [a Dell Optiplex GX-110 Pentium III 667 with 128 MB RAM], though it now boots off of a CD instead of the local hard disks to prevent any successful access from altering the operating system, installing a root kit, etc.; any and all changes will be lost when the firewall is rebooted. This is our network's primary defense, acting as the traffic cop deciding what traffic is and is not allowed. A second firewall, also running Netfilter on Red Hat 7.3, has been configured for the conference room access. An old Pentium II 350 PC with 64 MB RAM proved more than sufficient for this purpose.

The Citrix NFuse will still be the primary method for off-site staff to reach the network; a valid username and password, sent over an encrypted SSL connection using the RSA-SHA1 algorithm with a 128-bit key, will be required for this connection. (Smart card logon would be preferred, but has not been approved due to costs.) An L2TP over IPSec remote access solution is provided for off-site to synchronize Microsoft Outlook if necessary, and SSH-2 is available for IT staff to manage servers remotely. Non-staff do not require access to any of these VPN services. All three of these VPN servers are located behind the firewall, to leverage the firewall's filtering and logging capabilities, but are separated from the internal network to help isolate them.

Intrusion Detection Systems (IDS's) have been added to the service network and the internal network. In accordance with the "spend no money" mantra, this task is performed via Snort ([www.snort.org](http://www.snort.org)) running on Red Hat Linux 7.3. Decent PC's leftover from downsizing (Dell Optiplex GXi's with Pentium III 733's) have been repurposed for this task after the addition of memory and hard disk space. [Note that staff still using the 4-year old Pentium II 350's were not necessarily told all of these details...]

One important change that is not visible in this diagram is that all machines will have Symantec's Antivirus Corporate Edition<sup>7</sup> running, and will also have personal firewall software installed. All Linux machines will use the Netfilter/iptables solution for this firewalling; Windows 2000 machines will use Symantec's Desktop Firewall 2.0.<sup>8</sup> (The purchase of the Symantec products was approved because it was deemed too critical to put off until the next year's budget.) All personal firewalls will be configured to allow only the types of traffic that are required.

E-mail is still handled by Exchange 2000 & Outlook 2002, but because it is virtually impossible to "harden" an Exchange 2000 server due to the number of services running, all SMTP traffic will be routed through a Postfix mail relay located in the DMZ. (Note that this server will not be an "open relay;" it will be configured to relay mail only for the internal Exchange server.)

The existing hubs and switches, comprised of 3Com SuperStack II Switch 3000's and multiple 3Com "autosensing" SuperStack II 10/100 hubs, were reused; new switches, including Gigabit Ethernet switches for the backbone, have been budgeted for next year.

### ***Summary of new security policy***

- 1) All access levels will follow the principle of least privilege.
- 2) All Windows machines, whether servers or PC's, will have up-to-date antivirus software running at all times. This will be provided by Symantec's AntiVirus Corporate Edition.
- 3) All Windows machines will have Symantec's Desktop Firewall 2.0 installed and configured. This will be distributed and managed from a centralized server.
- 4) All Red Hat machines will take advantage of having Netfilter built into the Linux kernel, and will restrict all inbound and outbound traffic to only that which is required for the service(s) the machine provides.
- 5) The router connected to the Internet will be reconfigured to provide as much filtering as possible without degrading performance. (A replacement Cisco router with improved capabilities has been budgeted for the next fiscal year, and will be purchased and implemented as soon as possible.)
- 6) All e-mail will be routed through an SMTP relay server. This server will run Postfix Version 1.1 Patchlevel 11<sup>9</sup> on a Red Hat bastion host.
- 7) All e-mail, both inbound and outbound, will be scanned for viruses by both the Exchange 2000 Enterprise server and the Postfix SMTP relay server.

<sup>7</sup> <http://enterprisesecurity.symantec.com/content/ProductJump.cfm?Product=155&PID=12930860&EID=0>

<sup>8</sup> <http://enterprisesecurity.symantec.com/content/ProductJump.cfm?Product=36&PID=12930860&EID=0>

<sup>9</sup> <ftp://ftp.cis.fed.gov/pub/postfix/index.html>

- 8) Internal staff's Internet access will be restricted to Web traffic, and will be filtered through a Red Hat server running Squid (<http://www.squid-cache.org/>). Due to the lack of business need, staff will not have access to Instant Messaging, Usenet News, or peer-to-peer software.
- 9) Traffic among servers within the service network or between the service network and internal network will be secured via IPSec whenever possible. All authentication traffic, even if only within the internal network, will be encrypted.
- 10) All administrative work done over the network must be secured. This will be done by using SSH-2 ([www.openssh.org](http://www.openssh.org)), IPSec, or HTTPS/SSL.
- 11) All passwords will, at a minimum, adhere to Microsoft's complexity requirements. Staff will be trained, and routinely reminded, how to develop complex passwords they can remember. Passwords for administrative and service accounts will use non-printing ASCII characters. [Note that Windows 2000 enforces password requirements universally; this means adhering to our policy for these administrative and service accounts will be a manual endeavor, subject to human error...]
- 12) All servers, whether in the service network or the internal network, will be hardened to the greatest extent possible (with the understanding that for some, such as Exchange 2000, the greatest extent isn't very great.)
- 13) All patches, updates, etc. will be applied as promptly as feasible. Multiple services will be subscribed to, and Websites routinely monitored, to ensure IT staff are immediately aware of security flaws with software used. [Lists monitored will include, but not necessarily be limited to, the SANS newsletters (<http://www.sans.org/newlook/digests/SAC.htm>), Windows & .NET Magazine's Security Advisor newsletter (<http://secure.duke.com/nt/SecAdmin/index.cfm?Code=sawi251xna>), and the famous bugtraq (<http://online.securityfocus.com/cgi-bin/sfonline/subscribe.pl>).] Microsoft's Software Update Service will be used for all Windows 2000 machines to minimize Internet traffic and exposure.
- 14) Traveling staff will use Windows 2000's Encrypting File System (EFS) to help protect data in the event of hardware theft.
- 15) Audits of the network will be performed on a regular basis. Logs will be monitored continually. Centralized syslog servers, one in the DMZ and one in the internal network, will be configured to support this purpose. All Windows 2000 servers will use NTSyslog (<http://ntsyslog.sourceforge.net/>) to send their Application, Security, and System event logs to the syslog servers.
- 16) All business data transmitted over the Internet by staff, suppliers, or partners will be encrypted and/or digitally signed.
- 17) Supplier, partner, and staff identities on the Web will be controlled by both standard username & passwords and by digital certificates, issued (and revoked) by GIAC Enterprises.

- 18) Any unauthorized installation of software will result in severe penalties, up to and including dismissal and criminal prosecution.
- 19) All Microsoft Windows 2000 computers, with the exception of servers on the service network, will be part of the GIACENT.COM Active Directory domain (GIAC Enterprises' public domain name is GIAC.COM). Group Policy will be used to manage all of these Windows 2000 computers and, in particular, tighten their security. To the greatest extent possible, all Windows 2000 Professional machines will conform to the CIS Gold Standard ([http://csrc.nist.gov/itsec/guidance\\_W2Kpro.html](http://csrc.nist.gov/itsec/guidance_W2Kpro.html)).
- 20) Microsoft Internet Explorer and Outlook will both be locked down as much as possible via the use of Group Policies and registry edits.
- 21) A detailed Business Continuity Plan will be developed and regularly reviewed. This will include data backup and recovery procedures. Recent backups will be stored off-site in a secure location.
- 22) All servers and other critical network hardware will be located in a secured room, dedicated to that purpose. Access to this room will be restricted.
- 23) With the exception of the public interface of the VPN server, all machines behind the firewall will use private, non-routable IP addresses; the firewall will provide Network Address Translation (NAT) functionality.
- 24) No machines in the internal network will have direct access to the Internet. This includes the NTP server, which will synchronize with the NTP server in the service network; the Norton AntiVirus server, which will obtain its updates via passive FTP from the Norton AntiVirus server located in the service network; and the Software Update Services server, which will obtain its updates from the SUS server in the DMZ.
- 25) DNS used for Active Directory and the internal network will be "split brain" DNS. Only one of the internal DNS servers will have access to the service network's DNS server, which will serve as a caching DNS server. This DNS server in the DMZ will host no information about either the internal network or the DMZ. [Hosts files will be used on servers in the DMZ when needed.]
- 26) The publicly accessible, authoritative DNS server for the GIAC Enterprises' "giac.com" domain is currently hosted offsite. The IT Manager routinely checks the security of this machine as best he can; he has proposed bringing it in-house to better ensure its security and accuracy, but that move has not yet been approved.
- 27) Remote access will be provided primarily via Citrix NFuse and Citrix Secure Gateway. Any VPN requirements not met by NFuse will be provided by either SSH, where possible, or L2TP. Both inbound and outbound SSH will terminate at a designated SSH server on the service network; any further connections necessary will be made via SSH from this server only.
- 28) Old, unused, yet functional PC's will be reconfigured as Netfilter firewalls for home use by telecommuting staff; Pentium 133's with 64 MB RAM are currently being tested for this purpose, with great success.

These will have the ability to be monitored and managed by GIAC Enterprises' IT staff via SSH. Windows laptops used by staff will have Symantec's Desktop Firewall 2.0 installed.

- 29) All remote access via SSH or L2TP over IPSec must be done from machines with personal firewall software installed and configured.
- 30) All traffic to the MS SQL server will be secured via IPSec, whether from the Web server or the internal network.

© SANS Institute 2000 - 2002, Author retains full rights.

## Assignment 2: Security Policy & Tutorial

### Overview

There are three primary parts to implementing our new security policy. These are the border router, the firewall, and the VPN servers. We will outline how we use these to implement and enforce our policies defined above. In addition, our explanation of the border router will include a tutorial for configuration.

We will use our border router as our first line of defense, blocking invalid (i.e. spoofed) ip addresses and only allowing the types of traffic we want. This, combined with our firewall, is the most obvious form of defense in depth.

Our firewall will also be configured to block invalid addresses, allowing only the types of traffic our policy states we need. This will be our primary defense.

We will also implement Intrusion Detection Systems running Snort. We expect a lot of false positives, particularly at the outset; hopefully this will improve over time as we tweak our configuration.

We will also have 3 different methods of VPN access. The majority of remote access will be secured through the use of Citrix NFuse with Citrix Secure Gateway. "Road warriors" will be able to synchronize via an L2TP over IPsec connection. The third method will be SSH-2, for IT administrators only. The primary firewall will be leveraged for filtering and logging of all of this remote access activity.

Logging will be done via syslog for Red Hat machines and NTSyslog for Windows servers, with all logging in the perimeter going to a syslog server in the DMZ. This server will send it's info to a syslog server in the internal network.

Machines will use NTP in order to keep their times synchronized. This is critical not only for having sensible logs, but much of our communications, such as Kerberos, require it.

### Access requirements by component:

#### 1. Border Router

Our router just needs to allow the requisite traffic through. Inbound, this amounts to HTTP, HTTPS (SSL), and SMTP (e-mail) for everyone except our own remote staff, who also have access to the LT2P over IPsec VPN. Our IT staff also has access to SSH-2, so we'll let that in too. However, as this is just a



border router, performing no authentication for traffic going through it, we have to open up all that traffic to everyone.

For internal staff, the same traffic is allowed out, with the addition of FTP for IT staff -- though again, the router doesn't discriminate; it lets anyone inside use FTP. Our router also allows PPTP traffic to itself, for the purposes of remote maintenance by IT staff; this is only enabled on the LAN side of the router. Nobody else needs or should have access directly to the router, so they won't.

## 2. Primary Firewall

Our firewall, running Netfilter, will allow the same inbound traffic as the border router, though it also performs Destination NAT (DNAT) to redirect that inbound traffic to the appropriate server. This provides our customers, partners, off-site staff, and suppliers with the access they require to our Web-based services.

Outbound traffic, again, is the same as the router, though for any traffic generated in the DMZ or internal network, Source NAT'ing takes place so our traffic will work out on the Internet. Our L2TP over IPsec traffic is left un-altered, however (as it must be in order for IPsec to work.)

All traffic destined to or from the Internal network, aside from the authenticated L2TP VPN users, must be routed through the DMZ to get to or from the Internet.

As with the border router, direct access to the firewall is provided for GIAC's IT staff; no other users require direct access. This will be via SSH-2, and only from the DMZ.

## 3. Virtual Private Networking

As stated above, we have three methods of VPN. The overwhelming majority will be through Citrix NFuse with Citrix Secure Gateway; this combination allows staff to do the same work off-site that they do on-site, accessing all the same resources. This doesn't require any special firewall configuration on the outside; we will be using HTTPS for this traffic (though ICA and SSL traffic will be allowed from the DMZ to the Citrix MetaFrame servers on the internal network.)

The second VPN will be the L2TP for our traveling sales force. This will use digital certificates issued (and revoked) by GIAC to establish IPsec tunnels using 3DES (see the event log sample in the section on VPN's, [below](#).) Assuming their machines are authenticated, they will then have to log on using a username and password, which will be secured via MS-CHAP v2. Once they have successfully connected, they will be able to synchronize Microsoft Outlook -- but very little else. We don't expect this to be used very heavily.

The third VPN is SSH-2, via OpenSSH ([www.openssh.org](http://www.openssh.org)), for any off-site IT

staff. This will enable them to monitor and manage the servers even in the rare event the IT staff managed to escape from the office.

Non-staff do not require access to any of these 3 VPN methods, since they will do everything via the Web site.

These three components combine to provide the following access:

1. Customers, Suppliers, Partners, and indeed anyone else on the Internet can access the [www.giac.com](http://www.giac.com) public Web server; access to the various services on this machine will be controlled by IIS, digital certificates, and NTFS permissions. Everyone will also be able to access the logon screen for Citrix NFuse, though only users with valid usernames and passwords will be able to successfully log on. [Note that this presents a potential Denial of Service, as someone could lock out accounts by attempting brute force logins with a valid username! The intent in the near future is to improve this through the use of digital certificates.]
2. Off-site staff with valid digital certificates will be able to access the L2TP over IPSec and SSH-2 servers, and will also require valid usernames and passwords. They will also be able to log onto the Citrix NFuse site.
3. On-site employees will be able to access the Internet, but only ports 80 and 443, and only via a proxy; they will not be able to access anything on the Internet directly. IT staff will also have access to port 21 for passive FTP.
4. Potential clients will have access to the Internet when they are in the main conference room.

All three of these components -- the border router, the firewall, and the VPN's -- are described in much greater detail below. We will start with our tutorial on the border router.

## **1. The Border Router:**

### ***A Tutorial for Configuring the Netopia R5300***

We will begin our explanation of how to implement the security architecture defined in Assignment 1 above at the outer edge of the GIAC Enterprises network, at the router connected to the Internet. This connection is a T-1 line (1,544 Mbps).

This router is, unfortunately, a lowly Netopia R5300 (<http://www.netopia.com/equipment/pdf/spec/r5000.pdf>). While it has the ability to perform NAT, DHCP, and packet filtering, none of these capabilities had been taken advantage of. The machine only has 1 MB of memory, which means it probably lacks the brawn necessary to do everything fully without impacting its performance. Fortunately, we don't need it for NAT or DHCP. We will, though, implement its packet filtering capabilities as much as we can. Even though the filter is not stateful, the expectation was that we would have to limit how many filter sets we implemented; much to our surprise, however, performance was not

significantly degraded.

As a result, the router is being used as the first line of defense for filtering out any and all traffic with illegitimate source IP addresses. We use the IANA list, located at: <http://www.iana.org/assignments/ipv4-address-space>, to define “illegitimate.” [Note: the “public” IP addresses for GIAC Enterprises have been replaced within this text, for security reasons, to an address block that we classify as illegitimate. The exclusion of these illegitimate addresses from our rulesets blocking spoofed addresses, and their inclusion elsewhere, is for purposes of this text only. There are also reminders about this in the rulesets, which have required slight altering to permit our “example” addresses.]

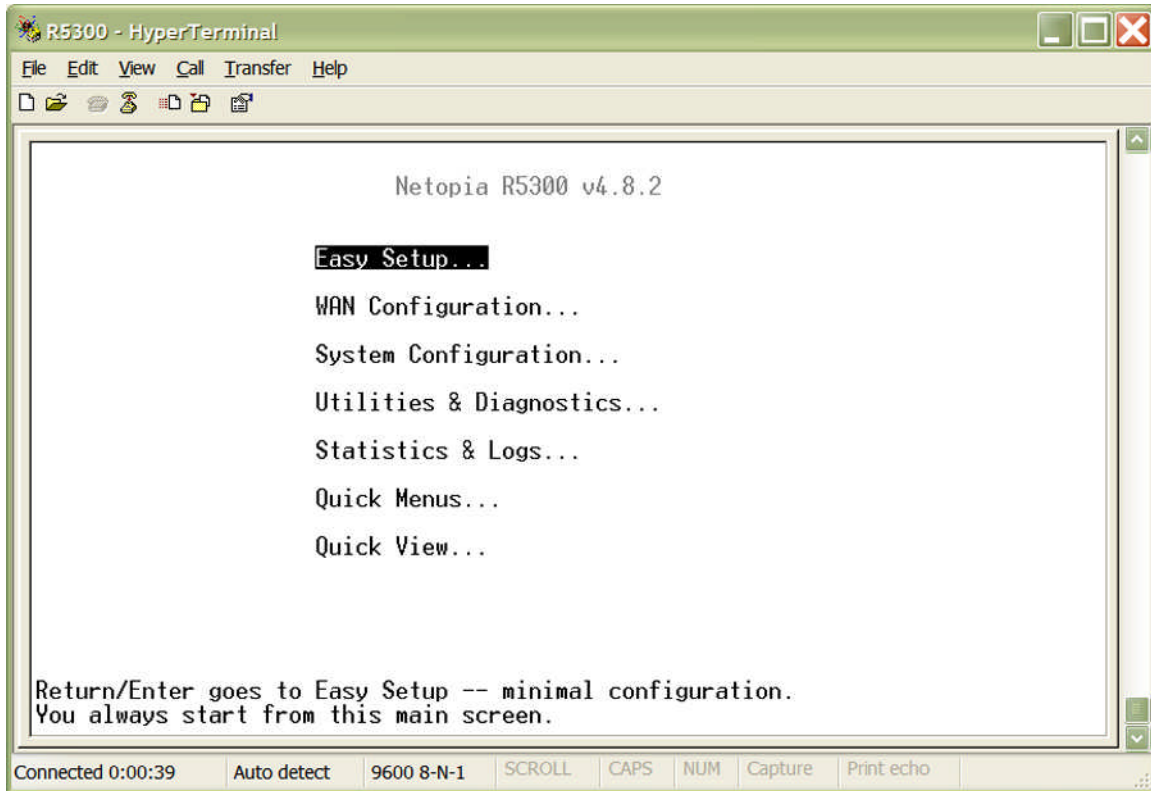
Configuring this router initially requires direct access via null modem cable. While the router has the capability for IP-based access, there was initially no encryption or security of any kind beyond a simple username and password, sent in plain-text. This was deemed unacceptable, particularly since the traffic is, by definition, outside the firewall; the IP access capability was therefore disabled until a more suitable option was available. That option -- PPTP -- turned out to be available via an existing firmware upgrade.

Implementing the changes that follow therefore entailed connecting a laptop to the router via null modem cable, starting the Windows 2000 HyperTerminal program (located under Start>Programs>Accessories>Communications>HyperTerminal), and configuring our settings as per page 6-4 in the Netopia User Reference Guide on the CD that accompanied the router or on the Internet at <http://www.netopia.com/equipment/pdf/manuals/r5000/leaseref.pdf>. Note, however, that the 56,700 baud settings recommended here may not work; I recommend 9600 baud. [The R5300 allows you to configure the connection speed at any range from 9600 to 56,700; however, only 9600 baud works reliably for us.] Another “gotcha” is that in practice, it is entirely possible that cycling the power on the router will be required to establish the connection! As this obviously breaks the connection with the Internet, it’s important to ensure this is only done during off-peak hours.

So, naturally, our work was done over the weekend, during the night, which is traditionally a period with little or no business activity. Customers, partners, and suppliers were all notified in advance that service could be interrupted during this time due to “network maintenance.”

Below we see the screen the Netopia R5300 provides us after we log in. (Those paying close attention will notice that, yes, this was done on a machine running Windows XP Professional rather than Windows 2000, even though we stated earlier that we were a fully Windows 2000 and Red Hat 7.3 shop. However, the Windows XP machine in question has since been downgraded to Windows 2000 for the sakes of both conformity and performance, and because the

Windows 2000 administrative MMC snapin tools do not function in Windows XP. But I digress.)



*Figure 3*

First, we confirm the IP access capability is turned off. We do this before anything and everything else in order to ensure we're the only ones working on the router, and to prevent someone else from altering any of our work.

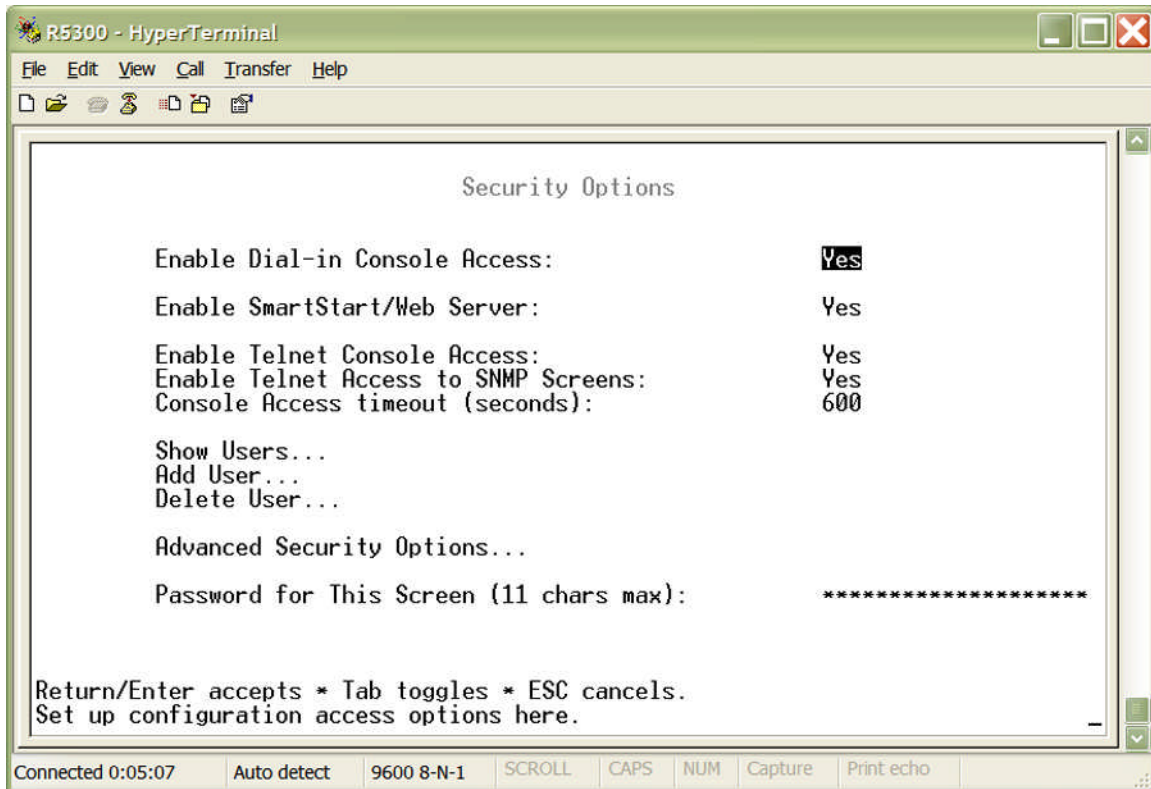


Figure 4

As you can see in *Figure 4* above, both telnet and Web access were enabled (this is the default setting.) Also note that as no filters have been applied at this point, anyone on the planet with an Internet connection was free to bang away attempting to guess the username & password as often and as long as they wanted. We changed the Web Server and Telnet options to “No.”

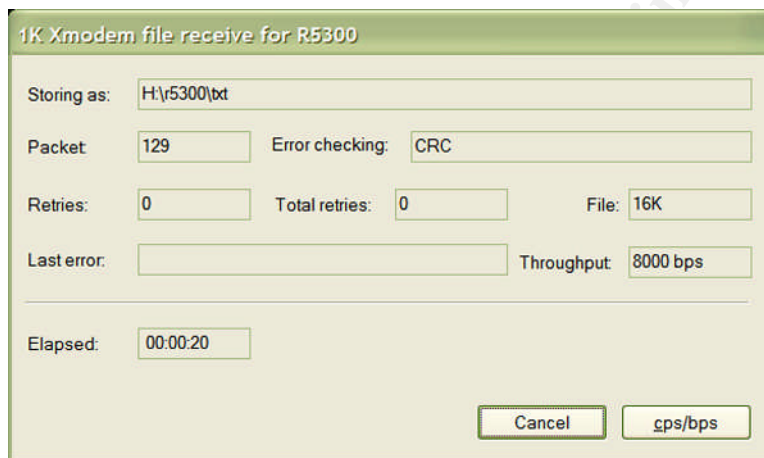
The next step after establishing the connection via serial cable was to download the router’s current configuration. Our plan was to edit it using our text editor of choice, and then upload our new configuration to the router. This would be vastly more efficient than the very tedious method of adding each ruleset one at a time using the console interface. *Figure 5* shows the process of downloading the router’s configuration via X-Modem transfer; note that with HyperTerminal, **contrary to the documentation**, we need to select “1k Xmodem” as the transfer protocol in order to get this to work. Once we were certain the file transfer process worked properly, we also uploaded updated firmware; this process had never been done with this equipment previously, but as part of GIAC Enterprises’ new policies, routine checking for firmware updates is now standard. The latest firmware can be acquired from [http://www.netopia.com/en-us/equipment/purchase/fmw\\_update.html](http://www.netopia.com/en-us/equipment/purchase/fmw_update.html) – the issues<sup>10</sup> addressed by this firmware upgrade (4.11 as of this writing) were numerous. The most important additions for us were the addition of PPTP<sup>11</sup> capability, the ability to designate

<sup>10</sup> See <http://www.netopia.com/en-us/equipment/upgrades/411Feat.html> for a list of changes

<sup>11</sup> Note that PPTP has been available since firmware version 4.4.

separate filter sets for the internal and external interfaces, and a text-formatted configuration dump. We will take advantage of the PPTP addition, as it enables us to turn the telnet over IP administrative functionality back on. Because our telnet will be within the PPTP tunnel, it should be secure enough for our purposes. (Note that as of firmware version 4.8, the R5300 can also use IPSec. However, this won't work for our purposes, because the machines we'd be trying to connect with are NAT'ed.)

With our firmware upgrade, we also now have a separate "superuser" account for configuring the WAN connection. This gives us an added level of security, slightly limiting the damage a remote connection can do.



*Figure 5, downloading the configuration*

Our plan to edit the configuration file off-line was foiled, however, by the fact that the configuration file we downloaded is in binary format. The good news is that this method can be used to create a backup of the router's configuration. The bad news is that, unfortunately, modifying the file is not possible, as far as we can tell. This means that we must resort to the glorious method of adding each of our rules one ruleset at a time via the console interface. We will spare you (and the trees, should you decide to print this) the zillions of screenshots involved in outlining that entire process, and instead simply show the process required to enter these entries. We will then list the completed rulesets.<sup>12</sup>

Next, we proceed to the meat of our activity, configuring the packet filtering. We will use the IANA list (see above) to block all unassigned or restricted addresses, then we will enable the traffic we want to allow. Everything else will be dropped. The majority of the work we'll be doing will be done under the System Configuration option shown above in *Figure 3*. *Figure 6* below shows the options on the System Configuration sub-menu.

<sup>12</sup> Our firmware upgrade actually added the ability to use text formatted configuration dumps, but unfortunately we didn't see that during our initial reading of the accompanying notes!

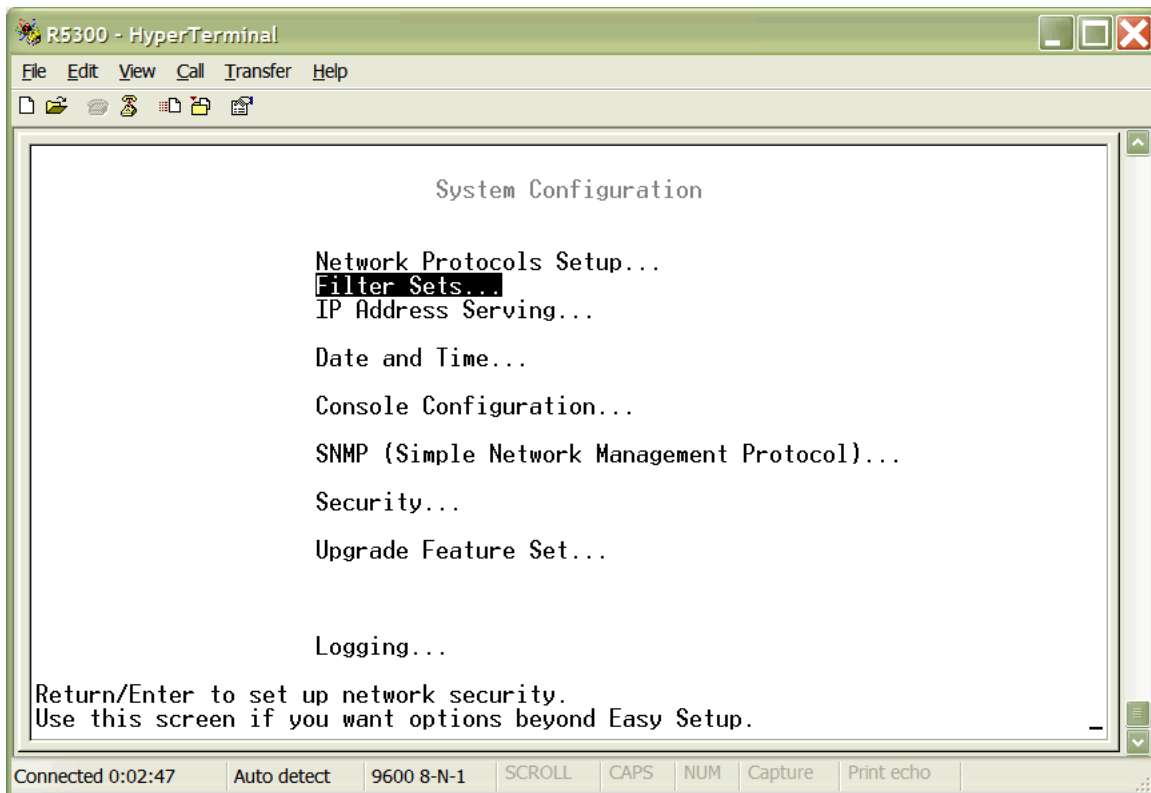
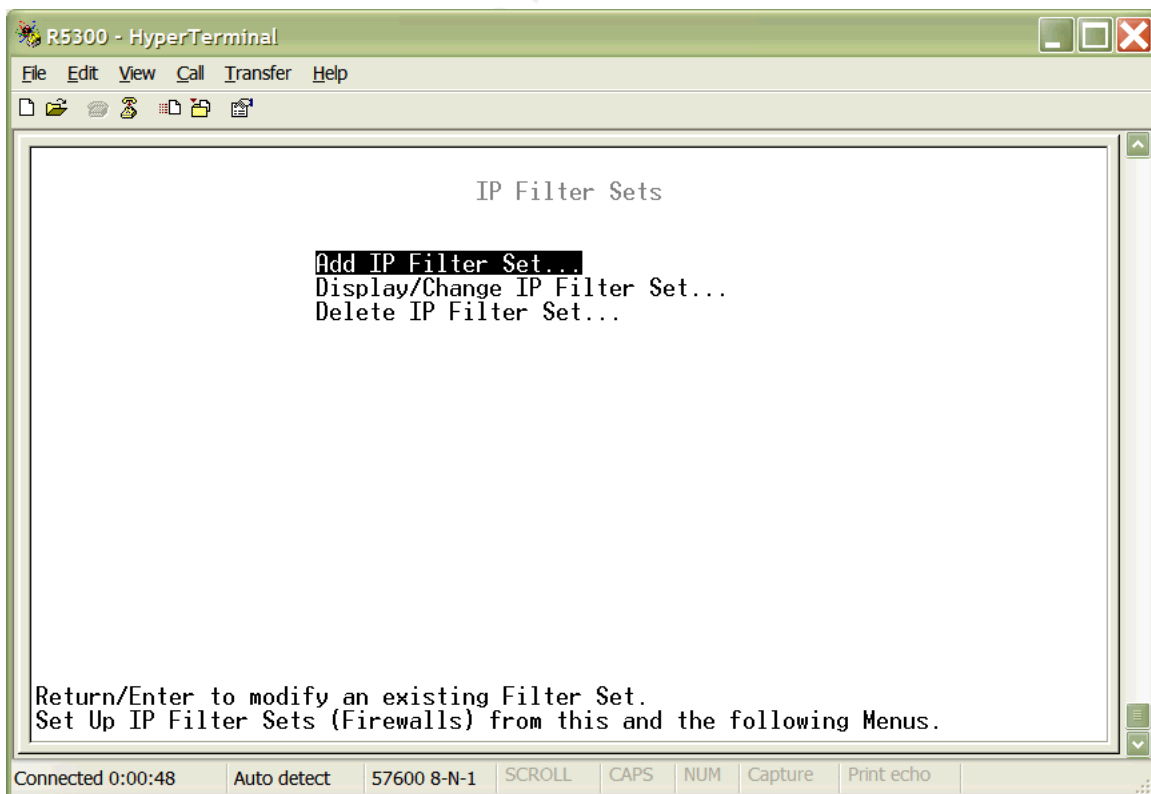


Figure 6



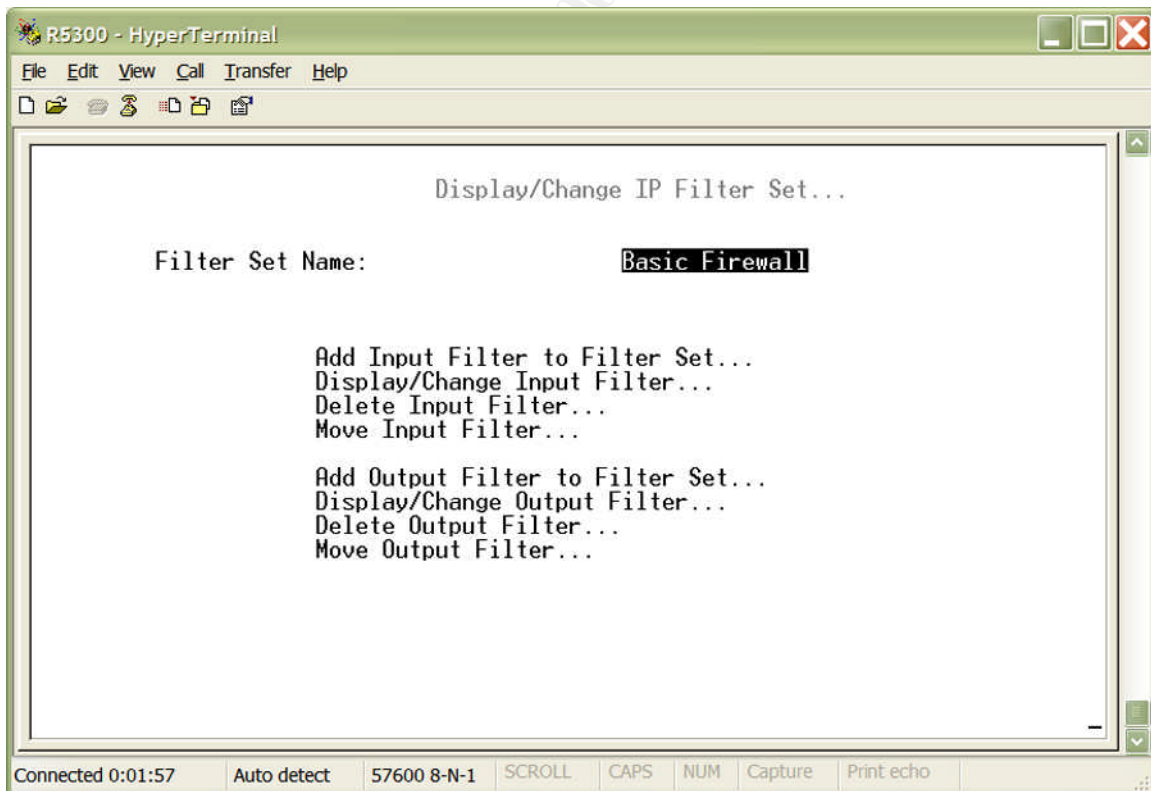


*Figure 7*

We had noticed previously that a “Basic Firewall” filter set already existed. Whether it was built-in or simply a test by the previous IT Manager is not known; in any event, before creating our own we decided to check it out. We found that it denies tcp ports 2000 and 6000 – but seems to allow pretty much everything else. We’ll make a new filter set that’s just a wee bit better.

As stated previously, the new firmware enables us to designate separate filter sets for the internal and external interfaces. This is a mixed blessing; we are still limited to a combined total of 255 rules in 8 filters. Of course, using them all would probably overwhelm the little machine anyway and negatively impact performance; we will therefore limit our “invalid” source address filtering to the external interface.

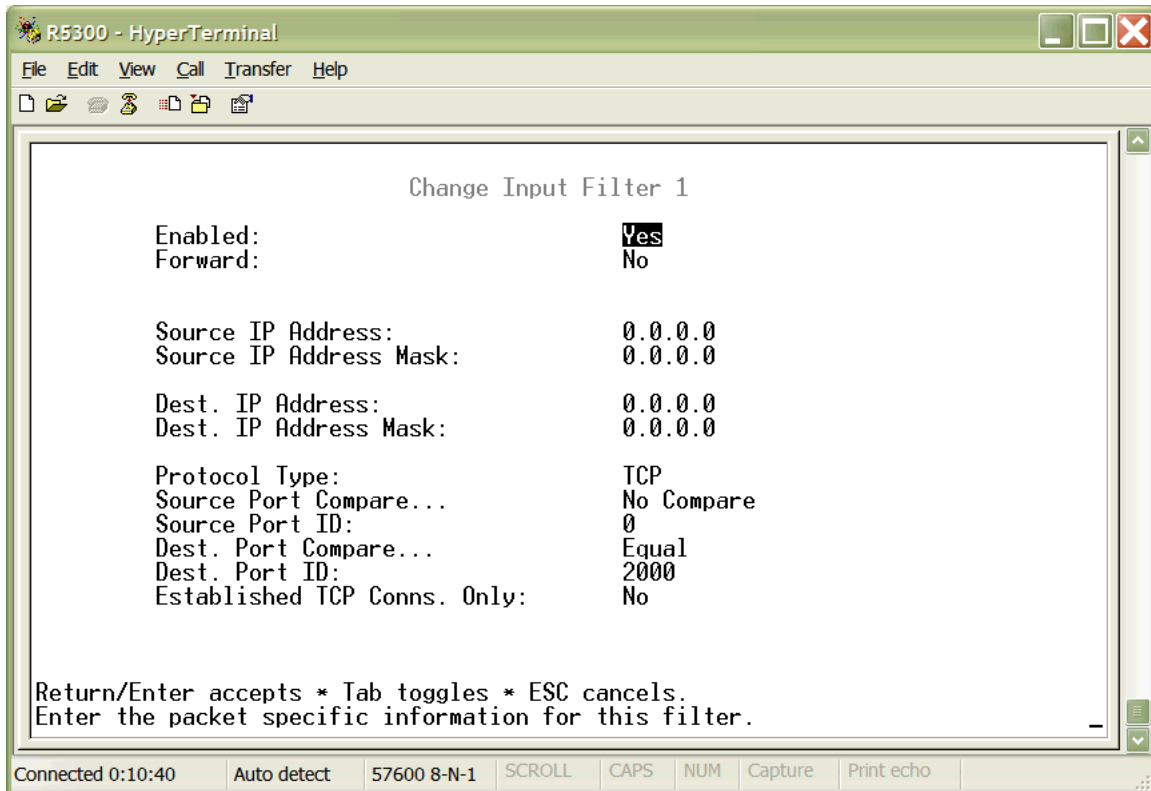
Note that the Netopia R5300 gives us Input and Output filters; for the external, Internet-facing interface, “Input” refers to traffic coming into the network, and “Output” refers to traffic going out to the Internet. This is reversed on the internal, LAN side interface; “Input” here refers to traffic coming into the router, and “Output” refers to traffic going to the internal network.



*Figure 8*



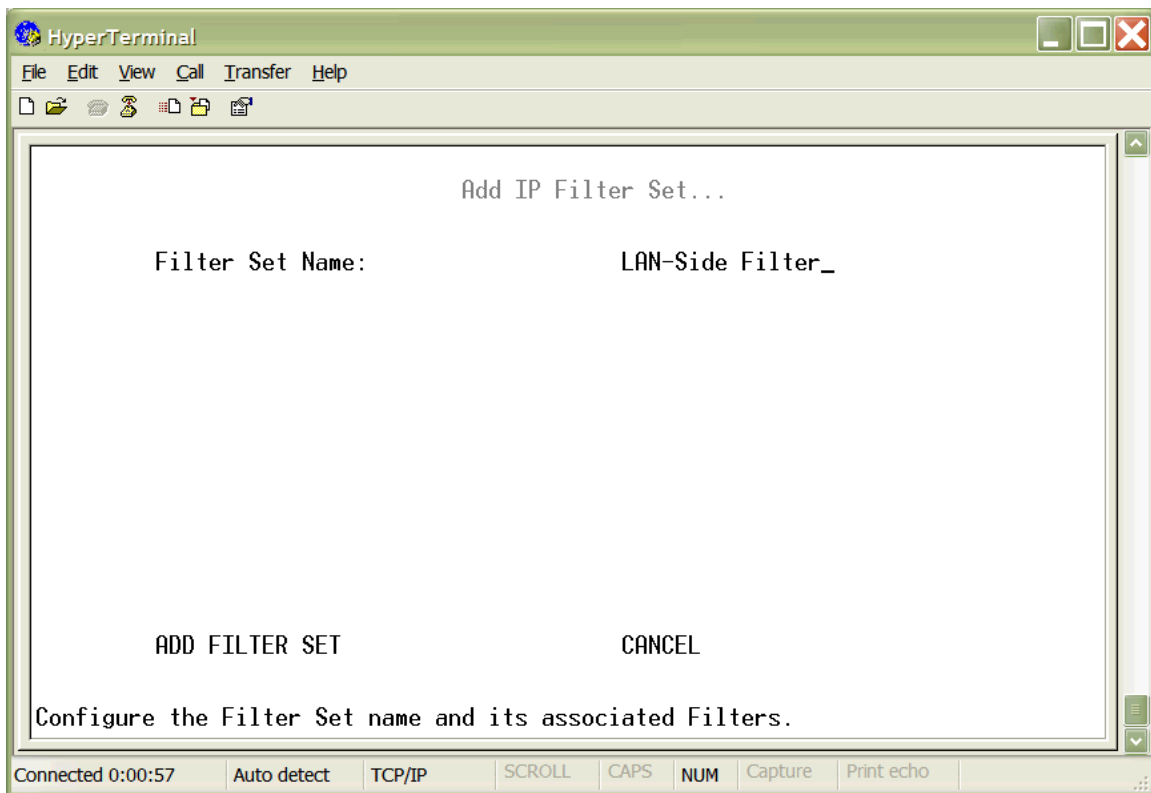
First, we select Display/Change Input Filter; *Figure 9* shows we could just use the included Basic Firewall by changing the rules.



*Figure 9*

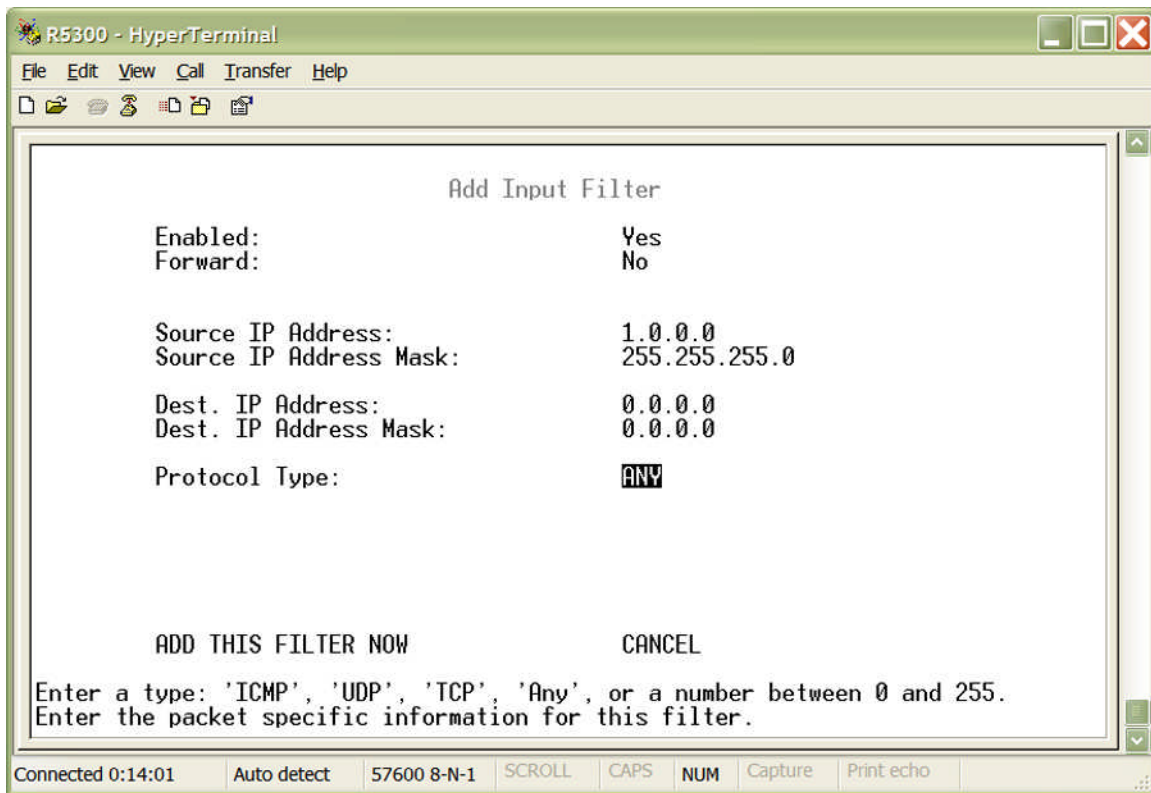
The Netopia R5300 firewalling capabilities are not stateful, meaning we can not keep track of whether or not a connection is “established.” (Given the router’s 1MB of RAM, this is probably a good thing! Otherwise, a Denial of Service attack designed to fill up the state table in memory used to track connections would be trivial.) We can designate protocol type, source and/or destination IP address, source and/or destination port (if relevant), whether or not the filter is active, and whether or not the filter should forward the packet. Rulesets are applied sequentially, from #1 on up, so order is important for both proper security and performance.

First, we created custom rulesets with friendly names. This is shown below, in *Figure 10*.



**Figure 10**

Next, we started by configuring our rulesets to block spoofed source addresses. This is shown in *Figure 11*.



*Figure 11*

After adding all of the IP addresses to check for – a very tedious process! – we added rulesets for the “Input” traffic we want to allow. The first of these is shown in *Figure 12* below.

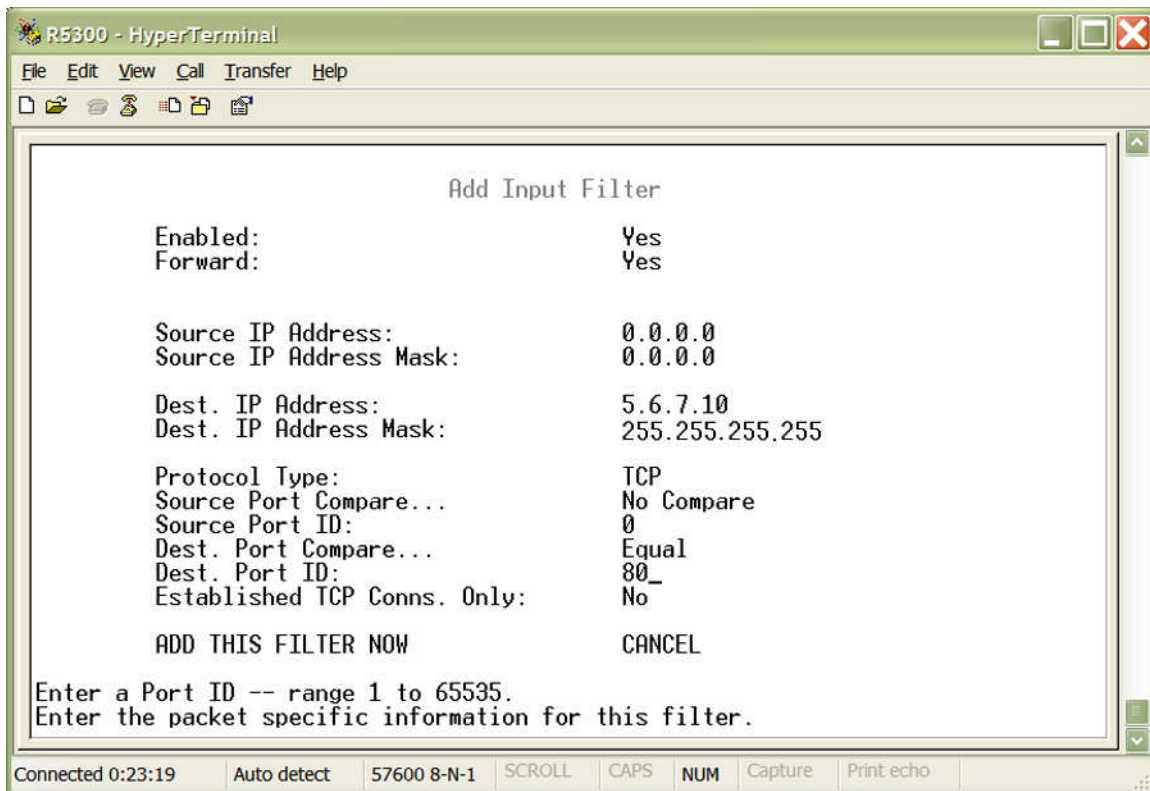


Figure 12

Note that while this screen shows an option for "Established TCP Conns. Only," this only checks for the ACK bit set. We will, however, take advantage of this where appropriate. (Note that this setting and the subnet mask are only visible on this screen; they will not show up in the screens listing our filters' rulesets.)

The only traffic generated by the router itself that we'll allow is syslog (UDP port 514) and replies to our PPTP sessions. As you can see in Figure 13 below, the Netopia R5300 does in fact have the capability of sending its logs to our centralized syslog server. There was much debate over whether or not to permit syslog traffic into the network from the router. If the router were to be compromised, or its filter set fail, we would essentially be opening up our syslog server, and hence all of our logging, to input from the outside world. After careful consideration of this risk, the decision was made to take advantage of the logging capabilities anyway, because the odds that staff would actually view and sort through the logs on the router manually each day are very slim. We will use the filter sets of both the router and the primary firewall to help ensure the incoming syslog traffic does indeed come from the router, understanding there is no way to truly secure this traffic. In the Input filter, we drop any traffic with a source IP address of the router; this prevents spoofing of our own address. In the LAN-side's Output filter, we allow only UDP port 514 from our router's IP address destined to our DMZ's syslog server, plus our PPTP replies. (The primary firewall itself will allow UDP port 514 only from our router's address

destined to our DMZ's syslog server. Note that while Netfilter can match based on MAC address, there's really no point in this since all traffic coming through the router will come from the router's internal MAC address.)

Note that we do, of course, want to log everything.

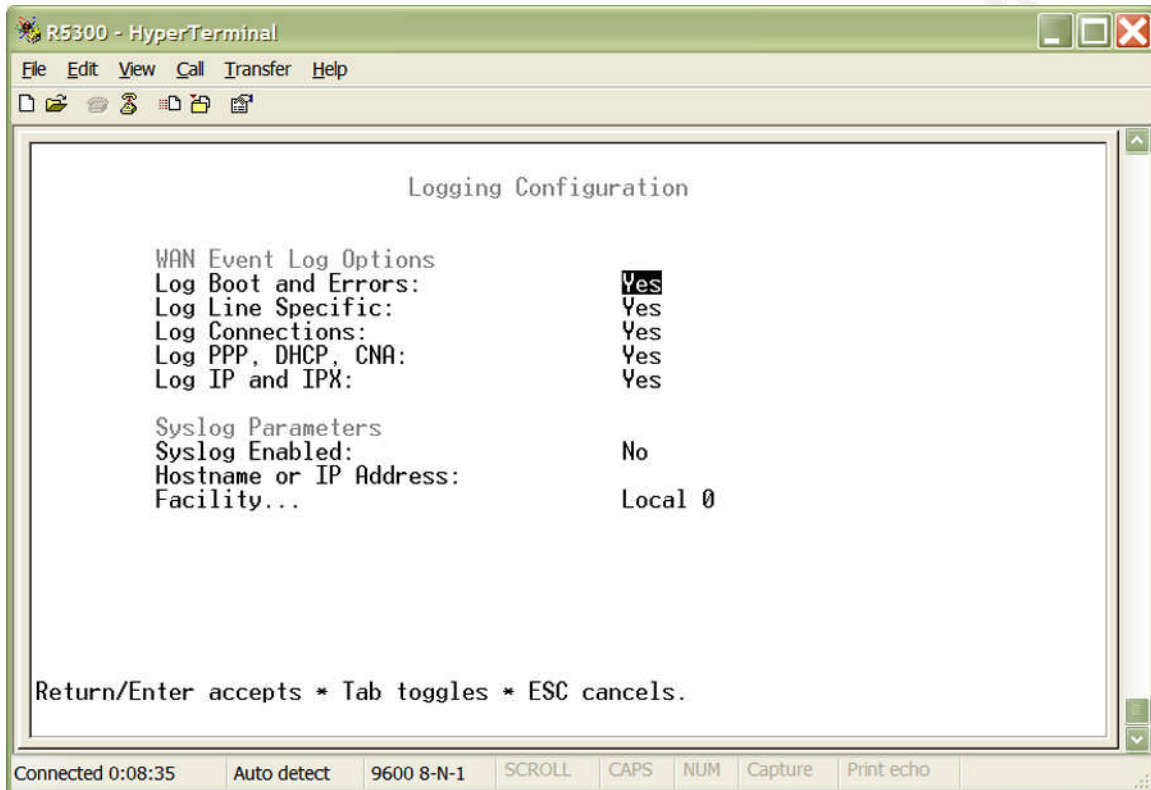


Figure 13, prior to firmware upgrade

As we see in *Figure 13* prior to our upgrade the syslog capabilities don't include logging the packet filter violations. This was not added until firmware version 4.10; since we're implementing 4.11, we can take advantage of it.

[There has been some philosophical discussion regarding whether or not it would be better to simply not use the router's filtering capabilities at all, enabling the traffic to reach the primary firewall, since Netfilter's significantly better logging capabilities would enable GIAC Enterprises to better analyze attempts at unauthorized access or other attacks. It was decided in the end, though, that the added layer of security to prevent security breaches was more important than the ability to watch those security breaches.]

At long last, here we see in *Figure 14* a screenshot of the configured router's rulesets. Note that we can only see the first 16 lines here.

#	Source IP Addr	Dest IP Addr	Proto	Src.Port	D.Port	On?	Fwd
1	1.0.0.0	0.0.0.0	ANY	--	--	Yes	No
2	2.0.0.0	0.0.0.0	ANY	--	--	Yes	No
3	7.0.0.0	0.0.0.0	ANY	--	--	Yes	No
4	23.0.0.0	0.0.0.0	ANY	--	--	Yes	No
5	27.0.0.0	0.0.0.0	ANY	--	--	Yes	No
6	31.0.0.0	0.0.0.0	ANY	--	--	Yes	No
7	36.0.0.0	0.0.0.0	ANY	--	--	Yes	No
8	37.0.0.0	0.0.0.0	ANY	--	--	Yes	No
9	39.0.0.0	0.0.0.0	ANY	--	--	Yes	No
10	41.0.0.0	0.0.0.0	ANY	--	--	Yes	No
11	42.0.0.0	0.0.0.0	ANY	--	--	Yes	No
12	49.0.0.0	0.0.0.0	ANY	--	--	Yes	No
13	50.0.0.0	0.0.0.0	ANY	--	--	Yes	No
14	58.0.0.0	0.0.0.0	ANY	--	--	Yes	No
15	59.0.0.0	0.0.0.0	ANY	--	--	Yes	No
16	60.0.0.0	0.0.0.0	ANY	--	--	Yes	No

...more...

Figure 14 (Normally we'd disallow 5.0.0.0 too)

As we mentioned previously, rather than multiple screenshots, we will simply insert the entire ruleset as a whole for legibility. We begin by dropping all traffic with an illegitimate source IP address. We then allow the inbound traffic we want, followed by the traffic we want to allow from our firewall out to the Internet. This allowed traffic will be either to or from the following addresses and ports:

**5.6.7.1** – This is the Router's IP address; we allow TCP 1723 (PPTP) and IP protocol 47 (GRE), in the Ethernet side Input filter (and return traffic on the Ethernet side's Output filter set.) We also allow UDP 514 (syslog) out the LAN interface to 5.6.7.12.

**5.6.7.7** – This is the address we use for our primary firewall's SNAT, i.e. all of the traffic generated by us from machines with private, non-routable IP addresses and headed out of the firewall to the Internet gets Source NAT'ed to this address. The only traffic we allow out to the Internet is destined to TCP ports 80, 443, 21 (ftp; we allow "passive" ftp only), 22 (SSH), 25 (SMTP) and UDP port 123 (NTP); these are also the only source ports we allow back to this address. Our Internet-bound traffic will go in the external interface's Output filter; rulesets for the replies to this traffic will go in the external interface's Input filter set.

**5.6.7.8** – This is the public address of our L2TP VPN server; we need to allow UDP ports 1701 (L2TP) and 500 (IKE), plus IP protocol 50 (ESP), inbound from the Internet (the external interface's Input filter.) Naturally, we also need to allow the replies, which will go in the Output filter set of the external interface.

**5.6.7.10** – This is the public address of our Web server; we allow TCP ports 80 (HTTP) and 443 (SSL). As this is also the public address of our SMTP server, we allow TCP port 25. This goes in the external interface's Input filter, with replies in the external Output filter.

**5.6.7.11** – This is the public address of our NFuse server; we allow TCP ports 80 and 443. This goes in the external interface's Input filter, with replies in the external Output filter.

**5.6.7.12** – This is the public address of our SSH and; we therefore need TCP port 22 (SSH). This goes in the external interface's Input filter, with replies in the external Output filter.

**5.6.7.13** – This is the address we use for our conference room's firewall (Firewall B) SNAT. Since we're providing services, we allow TCP ports 80, 443, 21 (ftp; we allow "passive" ftp only), 22 (SSH), and 1723 (PPTP); UDP ports 1701 (L2TP) and 500 (IKE); and IP protocols 50 and 47 (ESP and GRE, respectively). This all goes in the Ethernet side Input filter, with return traffic enabled in the Ethernet side's Output filter set.

**5.6.7.14** – This is the public address for Firewall B. We want outbound UDP 123 (NTP) for synchronization enabled in the Ethernet (LAN) side's Input filter, with the replies enabled in the Output filter.

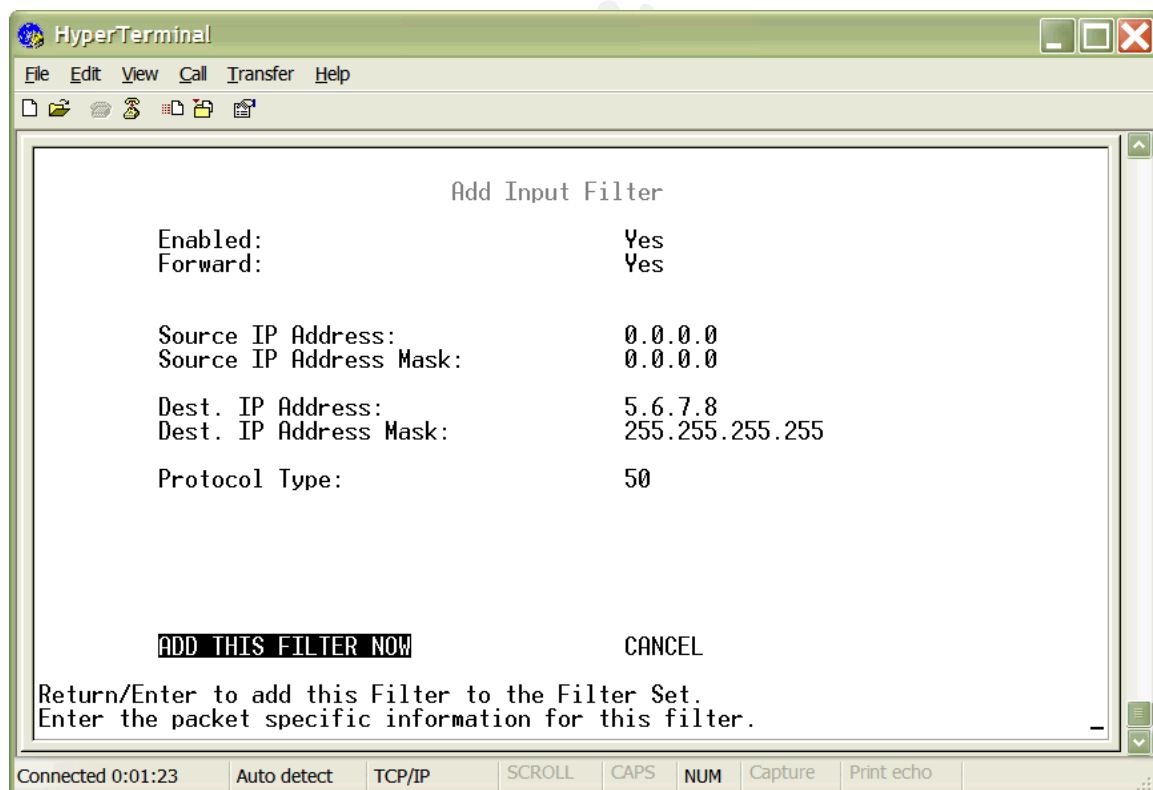


Figure 15, adding ESP

Here is the Netopia R5300 External Interface Input filter. Note that this router

determines what the default policy is, i.e. what to do with packets that don't match, based simply on what the settings for "Fwd" within the filter are. If the filter's rulesets all forward traffic, then packets that don't match will be dropped. If the filter's rulesets all drop traffic, then the packets that don't match will be forwarded. In the case of mixed rulesets, such as our Input filter which forwards some and drops others, packets that don't match are dropped.

Netopia R5300's External Interface's Input filter:

#	Source IP	Dest IP	Proto	Sport	Dport	On?	Fwd
1	1.0.0.0	0.0.0.0	ANY	--	--	Yes	No
2	2.0.0.0	0.0.0.0	ANY	--	--	Yes	No
3	3.0.0.0	0.0.0.0	ANY	--	--	Yes	No
4	7.0.0.0	0.0.0.0	ANY	--	--	Yes	No
5	23.0.0.0	0.0.0.0	ANY	--	--	Yes	No
6	27.0.0.0	0.0.0.0	ANY	--	--	Yes	No
7	31.0.0.0	0.0.0.0	ANY	--	--	Yes	No
8	36.0.0.0	0.0.0.0	ANY	--	--	Yes	No
9	37.0.0.0	0.0.0.0	ANY	--	--	Yes	No
10	39.0.0.0	0.0.0.0	ANY	--	--	Yes	No
11	41.0.0.0	0.0.0.0	ANY	--	--	Yes	No
12	42.0.0.0	0.0.0.0	ANY	--	--	Yes	No
13	49.0.0.0	0.0.0.0	ANY	--	--	Yes	No
14	50.0.0.0	0.0.0.0	ANY	--	--	Yes	No
15	58.0.0.0	0.0.0.0	ANY	--	--	Yes	No
16	59.0.0.0	0.0.0.0	ANY	--	--	Yes	No
17	60.0.0.0	0.0.0.0	ANY	--	--	Yes	No
18	70.0.0.0	0.0.0.0	ANY	--	--	Yes	No
19	71.0.0.0	0.0.0.0	ANY	--	--	Yes	No
20	72.0.0.0	0.0.0.0	ANY	--	--	Yes	No
21	73.0.0.0	0.0.0.0	ANY	--	--	Yes	No
22	74.0.0.0	0.0.0.0	ANY	--	--	Yes	No
23	75.0.0.0	0.0.0.0	ANY	--	--	Yes	No
24	76.0.0.0	0.0.0.0	ANY	--	--	Yes	No
25	77.0.0.0	0.0.0.0	ANY	--	--	Yes	No
26	78.0.0.0	0.0.0.0	ANY	--	--	Yes	No
27	79.0.0.0	0.0.0.0	ANY	--	--	Yes	No
28	82.0.0.0	0.0.0.0	ANY	--	--	Yes	No
29	83.0.0.0	0.0.0.0	ANY	--	--	Yes	No
30	84.0.0.0	0.0.0.0	ANY	--	--	Yes	No
31	85.0.0.0	0.0.0.0	ANY	--	--	Yes	No
32	86.0.0.0	0.0.0.0	ANY	--	--	Yes	No
33	87.0.0.0	0.0.0.0	ANY	--	--	Yes	No
34	88.0.0.0	0.0.0.0	ANY	--	--	Yes	No
35	89.0.0.0	0.0.0.0	ANY	--	--	Yes	No
36	90.0.0.0	0.0.0.0	ANY	--	--	Yes	No



37	91.0.0.0	0.0.0.0	ANY	--	--	Yes	No
38	92.0.0.0	0.0.0.0	ANY	--	--	Yes	No
39	93.0.0.0	0.0.0.0	ANY	--	--	Yes	No
40	94.0.0.0	0.0.0.0	ANY	--	--	Yes	No
41	95.0.0.0	0.0.0.0	ANY	--	--	Yes	No
42	96.0.0.0	0.0.0.0	ANY	--	--	Yes	No
43	97.0.0.0	0.0.0.0	ANY	--	--	Yes	No
44	98.0.0.0	0.0.0.0	ANY	--	--	Yes	No
45	99.0.0.0	0.0.0.0	ANY	--	--	Yes	No
46	100.0.0.0	0.0.0.0	ANY	--	--	Yes	No
47	101.0.0.0	0.0.0.0	ANY	--	--	Yes	No
48	102.0.0.0	0.0.0.0	ANY	--	--	Yes	No
49	103.0.0.0	0.0.0.0	ANY	--	--	Yes	No
50	104.0.0.0	0.0.0.0	ANY	--	--	Yes	No
51	105.0.0.0	0.0.0.0	ANY	--	--	Yes	No
52	106.0.0.0	0.0.0.0	ANY	--	--	Yes	No
53	107.0.0.0	0.0.0.0	ANY	--	--	Yes	No
54	108.0.0.0	0.0.0.0	ANY	--	--	Yes	No
55	109.0.0.0	0.0.0.0	ANY	--	--	Yes	No
56	110.0.0.0	0.0.0.0	ANY	--	--	Yes	No
57	111.0.0.0	0.0.0.0	ANY	--	--	Yes	No
58	112.0.0.0	0.0.0.0	ANY	--	--	Yes	No
59	113.0.0.0	0.0.0.0	ANY	--	--	Yes	No
60	114.0.0.0	0.0.0.0	ANY	--	--	Yes	No
61	115.0.0.0	0.0.0.0	ANY	--	--	Yes	No
62	116.0.0.0	0.0.0.0	ANY	--	--	Yes	No
63	117.0.0.0	0.0.0.0	ANY	--	--	Yes	No
64	118.0.0.0	0.0.0.0	ANY	--	--	Yes	No
65	119.0.0.0	0.0.0.0	ANY	--	--	Yes	No
66	120.0.0.0	0.0.0.0	ANY	--	--	Yes	No
67	121.0.0.0	0.0.0.0	ANY	--	--	Yes	No
68	122.0.0.0	0.0.0.0	ANY	--	--	Yes	No
69	123.0.0.0	0.0.0.0	ANY	--	--	Yes	No
70	124.0.0.0	0.0.0.0	ANY	--	--	Yes	No
71	125.0.0.0	0.0.0.0	ANY	--	--	Yes	No
72	126.0.0.0	0.0.0.0	ANY	--	--	Yes	No
73	172.16.0.0	0.0.0.0	ANY	--	--	Yes	No
74	172.17.0.0	0.0.0.0	ANY	--	--	Yes	No
75	172.18.0.0	0.0.0.0	ANY	--	--	Yes	No
76	172.19.0.0	0.0.0.0	ANY	--	--	Yes	No
77	172.20.0.0	0.0.0.0	ANY	--	--	Yes	No
78	172.21.0.0	0.0.0.0	ANY	--	--	Yes	No
79	172.22.0.0	0.0.0.0	ANY	--	--	Yes	No
80	172.23.0.0	0.0.0.0	ANY	--	--	Yes	No
81	172.24.0.0	0.0.0.0	ANY	--	--	Yes	No
82	172.25.0.0	0.0.0.0	ANY	--	--	Yes	No

83	172.26.0.0	0.0.0.0	ANY	--	--	Yes	No
84	172.27.0.0	0.0.0.0	ANY	--	--	Yes	No
85	172.28.0.0	0.0.0.0	ANY	--	--	Yes	No
86	172.29.0.0	0.0.0.0	ANY	--	--	Yes	No
87	172.30.0.0	0.0.0.0	ANY	--	--	Yes	No
88	172.31.0.0	0.0.0.0	ANY	--	--	Yes	No
89	172.32.0.0	0.0.0.0	ANY	--	--	Yes	No
90	172.33.0.0	0.0.0.0	ANY	--	--	Yes	No
91	192.168.0.0	0.0.0.0	ANY	--	--	Yes	No
92	197.0.0.0	0.0.0.0	ANY	--	--	Yes	No
93	222.0.0.0	0.0.0.0	ANY	--	--	Yes	No
94	223.0.0.0	0.0.0.0	ANY	--	--	Yes	No
95	240.0.0.0	0.0.0.0	ANY	--	--	Yes	No
96	241.0.0.0	0.0.0.0	ANY	--	--	Yes	No
97	242.0.0.0	0.0.0.0	ANY	--	--	Yes	No
98	243.0.0.0	0.0.0.0	ANY	--	--	Yes	No
99	244.0.0.0	0.0.0.0	ANY	--	--	Yes	No
100	245.0.0.0	0.0.0.0	ANY	--	--	Yes	No
101	246.0.0.0	0.0.0.0	ANY	--	--	Yes	No
102	247.0.0.0	0.0.0.0	ANY	--	--	Yes	No
103	248.0.0.0	0.0.0.0	ANY	--	--	Yes	No
104	249.0.0.0	0.0.0.0	ANY	--	--	Yes	No
105	250.0.0.0	0.0.0.0	ANY	--	--	Yes	No
106	251.0.0.0	0.0.0.0	ANY	--	--	Yes	No
107	252.0.0.0	0.0.0.0	ANY	--	--	Yes	No
108	253.0.0.0	0.0.0.0	ANY	--	--	Yes	No
109	254.0.0.0	0.0.0.0	ANY	--	--	Yes	No
110	255.0.0.0	0.0.0.0	ANY	--	--	Yes	No
111	0.0.0.0	5.6.7.7	TCP	80	--	Yes	Yes
112	0.0.0.0	5.6.7.7	TCP	443	--	Yes	Yes
113	0.0.0.0	5.6.7.7	TCP	21	--	Yes	Yes
114	0.0.0.0	5.6.7.7	TCP	22	--	Yes	Yes
115	0.0.0.0	5.6.7.7	TCP	25	--	Yes	Yes
116	0.0.0.0	5.6.7.10	TCP	--	80	Yes	Yes
117	0.0.0.0	5.6.7.10	TCP	--	443	Yes	Yes
118	0.0.0.0	5.6.7.10	TCP	--	25	Yes	Yes
119	0.0.0.0	5.6.7.11	TCP	--	80	Yes	Yes
120	0.0.0.0	5.6.7.11	TCP	--	443	Yes	Yes
121	0.0.0.0	5.6.7.8	UDP	--	1701	Yes	Yes
122	0.0.0.0	5.6.7.8	UDP	--	500	Yes	Yes
123	0.0.0.0	5.6.7.8	ESP	--	--	Yes	Yes
124	0.0.0.0	5.6.7.12	TCP	--	22	Yes	Yes
125	5.127.25.23	5.6.7.7	UDP	123	--	Yes	Yes
126	5.118.34.135	5.6.7.7	UDP	123	--	Yes	Yes
127	0.0.0.0	5.6.7.13	TCP	80	--	Yes	Yes
128	0.0.0.0	5.6.7.13	TCP	443	--	Yes	Yes

129	0.0.0.0	5.6.7.13	TCP	21	--	Yes	Yes
130	0.0.0.0	5.6.7.13	TCP	22	--	Yes	Yes
131	0.0.0.0	5.6.7.13	TCP	1723	--	Yes	Yes
132	0.0.0.0	5.6.7.13	GRE	--	--	Yes	Yes
133	5.127.25.23	5.6.7.14	UDP	123	--	Yes	Yes
134	5.118.34.135	5.6.7.14	UDP	123	--	Yes	Yes

The Output filter for the external interface:

1	5.6.7.7	0.0.0.0	TCP	--	80	Yes	Yes
2	5.6.7.7	0.0.0.0	TCP	--	443	Yes	Yes
3	5.6.7.7	0.0.0.0	TCP	--	21	Yes	Yes
4	5.6.7.7	0.0.0.0	TCP	--	22	Yes	Yes
5	5.6.7.7	0.0.0.0	TCP	--	25	Yes	Yes
6	5.6.7.10	0.0.0.0	TCP	80	--	Yes	Yes
7	5.6.7.10	0.0.0.0	TCP	443	--	Yes	Yes
8	5.6.7.10	0.0.0.0	TCP	25	--	Yes	Yes
9	5.6.7.11	0.0.0.0	TCP	80	--	Yes	Yes
10	5.6.7.11	0.0.0.0	TCP	443	--	Yes	Yes
11	5.6.7.8	0.0.0.0	UDP	1701	--	Yes	Yes
12	5.6.7.8	0.0.0.0	UDP	500	--	Yes	Yes
13	5.6.7.8	0.0.0.0	ESP	--	--	Yes	Yes
14	5.6.7.12	0.0.0.0	TCP	22	--	Yes	Yes
15	5.6.7.7	5.127.25.23	UDP	--	123	Yes	Yes
16	5.6.7.7	5.118.34.135	UDP	--	123	Yes	Yes
17	5.6.7.9	0.0.0.0	ANY	--	--	Yes	No
18	5.6.7.7	5.6.7.1	GRE	--	--	Yes	Yes
19	5.6.7.13	0.0.0.0	TCP	--	80	Yes	Yes
20	5.6.7.13	0.0.0.0	TCP	--	443	Yes	Yes
21	5.6.7.13	0.0.0.0	TCP	--	21	Yes	Yes
22	5.6.7.13	0.0.0.0	TCP	--	22	Yes	Yes
23	5.6.7.13	0.0.0.0	GRE	--	--	Yes	Yes
24	5.6.7.14	5.127.25.23	UDP	--	123	Yes	Yes
25	5.6.7.14	5.118.34.135	UDP	--	123	Yes	Yes

The LAN side Interface's Input filter; this is almost the same as the external interface's Output filter, except for the addition of PPTP:

1	5.6.7.7	0.0.0.0	TCP	--	80	Yes	Yes
2	5.6.7.7	0.0.0.0	TCP	--	443	Yes	Yes
3	5.6.7.7	0.0.0.0	TCP	--	21	Yes	Yes
4	5.6.7.7	0.0.0.0	TCP	--	22	Yes	Yes
5	5.6.7.7	0.0.0.0	TCP	--	25	Yes	Yes
6	5.6.7.10	0.0.0.0	TCP	80	--	Yes	Yes
7	5.6.7.10	0.0.0.0	TCP	443	--	Yes	Yes
8	5.6.7.10	0.0.0.0	TCP	25	--	Yes	Yes

9	5.6.7.11	0.0.0.0	TCP	80	--	Yes	Yes
10	5.6.7.11	0.0.0.0	TCP	443	--	Yes	Yes
11	5.6.7.8	0.0.0.0	UDP	1701	--	Yes	Yes
12	5.6.7.8	0.0.0.0	UDP	500	--	Yes	Yes
13	5.6.7.8	0.0.0.0	ESP	--	--	Yes	Yes
14	5.6.7.12	0.0.0.0	TCP	22	--	Yes	Yes
15	5.6.7.7	5.127.25.23	UDP	--	123	Yes	Yes
16	5.6.7.7	5.118.34.135	UDP	--	123	Yes	Yes
17	5.6.7.9	0.0.0.0	ANY	--	--	Yes	No
18	5.6.7.7	5.6.7.1	GRE	--	--	Yes	Yes
19	5.6.7.13	0.0.0.0	TCP	--	80	Yes	Yes
20	5.6.7.13	0.0.0.0	TCP	--	443	Yes	Yes
21	5.6.7.13	0.0.0.0	TCP	--	21	Yes	Yes
22	5.6.7.13	0.0.0.0	TCP	--	22	Yes	Yes
23	5.6.7.13	0.0.0.0	TCP	--	1723	Yes	Yes
24	5.6.7.13	0.0.0.0	GRE	--	--	Yes	Yes
25	5.6.7.14	5.127.25.23	UDP	--	123	Yes	Yes
26	5.6.7.14	5.118.34.135	UDP	--	123	Yes	Yes

Here is our Netopia R5300's internal, LAN interface's Output filter; with the exception of the additions of syslog and PPTP, this is similar to the External interface's Input filter:

1	5.6.7.1	5.6.7.12	UDP	--	514	Yes	Yes
2	5.6.7.1	5.6.7.13	GRE	--	--	Yes	Yes
3	0.0.0.0	5.6.7.7	TCP	80	--	Yes	Yes
4	0.0.0.0	5.6.7.7	TCP	443	--	Yes	Yes
5	0.0.0.0	5.6.7.7	TCP	21	--	Yes	Yes
6	0.0.0.0	5.6.7.7	TCP	22	--	Yes	Yes
7	0.0.0.0	5.6.7.7	TCP	25	--	Yes	Yes
8	0.0.0.0	5.6.7.10	TCP	--	80	Yes	Yes
9	0.0.0.0	5.6.7.10	TCP	--	443	Yes	Yes
10	0.0.0.0	5.6.7.10	TCP	--	25	Yes	Yes
11	0.0.0.0	5.6.7.11	TCP	--	80	Yes	Yes
12	0.0.0.0	5.6.7.11	TCP	--	443	Yes	Yes
13	0.0.0.0	5.6.7.8	UDP	--	1701	Yes	Yes
14	0.0.0.0	5.6.7.8	UDP	--	500	Yes	Yes
15	0.0.0.0	5.6.7.8	ESP	--	--	Yes	Yes
16	0.0.0.0	5.6.7.12	TCP	--	22	Yes	Yes
17	5.127.25.23	5.6.7.7	UDP	123	--	Yes	Yes
18	5.118.34.135	5.6.7.7	UDP	123	--	Yes	Yes
19	0.0.0.0	5.6.7.13	TCP	80	--	Yes	Yes
20	0.0.0.0	5.6.7.13	TCP	443	--	Yes	Yes
21	0.0.0.0	5.6.7.13	TCP	21	--	Yes	Yes
22	0.0.0.0	5.6.7.13	TCP	22	--	Yes	Yes

23	0.0.0.0	5.6.7.13	TCP	1723	--	Yes	Yes
24	0.0.0.0	5.6.7.13	GRE	--	--	Yes	Yes
25	5.127.25.23	5.6.7.14	UDP	123	--	Yes	Yes
26	5.118.34.135	5.6.7.14	UDP	123	--	Yes	Yes

This gives us a total of 211 rulesets in 4 filters, well below our 255 rulesets in 8 filters limit.

## 2. The Primary Firewall

Connected directly to the Netopia router are two firewalls. Both are hardened Red Hat 7.3 boxes running Netfilter & iptables; Firewall A is the primary firewall, Firewall B is for providing Internet access to guests in the conference room.

Before continuing, we should address the frequent confusion regarding the terminology here. The terms Netfilter and iptables are often used interchangeably, but this is inaccurate. The best explanation comes from the [www.netfilter.org](http://www.netfilter.org) site itself:

netfilter is a set of hooks inside the linux 2.4.x kernel's network stack which allows kernel modules to register callback functions called every time a network packet traverses one of those hooks.  
iptables is a generic table structure for the definition of rulesets. Each rule within an IP table consists out of a number of classifiers (matches) and one connected action (target).  
netfilter, iptables and the connection tracking as well as the NAT subsystems together build the whole framework.

(Taken from <http://www.netfilter.org/documentation/index.html#whatis>, September 14, 2002). You could also think of Netfilter as the frame the iptables rules are attached to. Without the iptables userland command, Netfilter isn't really doing anything for us; without Netfilter, there's nothing for iptables to work with. For simplicity's sake, unless expressly described otherwise, references in this text to Netfilter can generally be assumed to refer to the combination of Netfilter and iptables.

We'll deal with Firewall A first.

### A. Firewall A:

Because we have fairly low traffic demands, we can get away with using an older PC for this role. Because it was sufficiently running the previous firewall software on Windows NT, we anticipated the same hardware would be not only sufficient, but perform better, as a streamlined RH box. We were quite pleased with the performance from our lowly Dell OptiPlex GX-110, a Pentium III 667 with 128 MB RAM. We've added 3 additional NIC's in order to provide an interface for the DMZ, and 2 interfaces to connect the Windows 2000 RRAS VPN server. Fortunately for us, this machine is the "mini-tower" model, with one

on-board NIC, so it had enough PCI slots for us to add all the NIC's we wanted. 5 NIC's is not possible on a lot of PC's! As an added bonus, the box is just small enough to fit on its side on a shelf in our server cabinet.

One advantage to using existing PC hardware in this manner is that we have multiple PC's available to use, left over from last year's annual PC upgrade, because the overworked network admin never got around to finding new homes for the old equipment. This enables us to configure and test on the hardware in question without disrupting services. It also means that if our firewall PC fails, we'll have parts readily available for immediate replacement. Most importantly, as far as the Board of Directors was concerned, no additional equipment purchases were necessary.

The first step was to install Red Hat Linux 7.3 onto the machine. A fairly minimal installation was performed. While a full-blown Linux installation tutorial is outside the scope of this document, we will touch on a few of the key points here. Note that once we have things configured as we want, we will make a bootable CD, pull out the hard drive, and go from there. Doing the initial configuration on hardware identical to that which will be used helps us minimize the headaches involved with compiling the kernel and making the bootable CD, and generally eases testing.

One important aspect of Red Hat installation we should point out is the prompt to select the type of firewall to install. This is a definite "gotcha." During installation, Red Hat offers choices of Medium Security, High Security, Custom Security, or No Firewall. Though it seems counter-intuitive, the correct answer – even when, as in our case, the machine is only going to be a firewall – is "No Firewall." This is because Red Hat uses the outdated ipchains stateless packet filter by default, rather than the much-improved Netfilter/iptables. Because the two are incompatible, choosing anything other than "No Firewall" will require you to deal with removing ipchains before you can configure and use Netfilter.

Also note that the Netfilter version with RH 7.3 is not the latest, so we will download the current version (1.2.7a as of this writing) from <http://www.netfilter.org/download>. As Netfilter is part of the Linux kernel, this is not simply an RPM we can install; we must not only build Netfilter from source, but have to rebuild our kernel as well. This is a good thing, as it forces us to take the time to harden and tune our kernel. Also note that there are multiple additional patches and modules available in Netfilter's "Patch-o-Matic" (POM) system, which is a separate download. We will be using a couple of modules from POM.

There are of course other configuration steps required, as we need to "harden" this machine – removing unnecessary services, configuring Tripwire ([www.tripwire.com](http://www.tripwire.com)), etc. [Note that if we configure Tripwire at this stage, it will bark at us when we make the changes required to implement our firewall.

That's OK, we'll be able to update Tripwire's settings.] A detailed explanation of all the steps required to harden a Linux machine is beyond the scope of this document, however.

As explained above, we must compile our own kernel. Instructions for just how to do this are outside the scope of this document too; however, we do need to mention the modules related to Netfilter that we're including. Because we know what our needs are, we're just compiling most of them into the kernel rather than adding them as modules via insmod later on. Some, however, we'll include as modules either because we have to or we just want them around to apply if something comes up and we need it. We need the following in our kernel:<sup>13</sup>

CONFIG\_PACKET - not really Netfilter-specific, but needed

CONFIG\_NETFILTER - self explanatory...

CONFIG\_IP\_NF\_CONNTRACK - for keeping track of connections, i.e.

NAT

CONFIG\_IP\_NF\_FTP - for ftp connection tracking; we're adding this as a module, even though we're using passive ftp; just in case an emergency crops up, it's there if we need it

CONFIG\_IP\_NF\_IPTABLES - for filtering and NAT; our firewall won't do much without it...

CONFIG\_IP\_NF\_MATCH\_LIMIT - we use this to aid in preventing DoS attacks

CONFIG\_IP\_NF\_MATCH\_MULTIPORT - for matching packets based on a range of destination or source ports.

CONFIG\_IP\_NF\_MATCH\_TOS - for matching Type of Service field, obviously. We're not doing this currently, but may want to; it's in as a module.

CONFIG\_IP\_NF\_MATCH\_STATE - this is what makes it a "stateful" firewall.

CONFIG\_IP\_NF\_FILTER - self explanatory. Pointless without it...

CONFIG\_IP\_NF\_TARGET\_REJECT - allows us to reject packets with ICMP or TCP RST. Very nice.

CONFIG\_IP\_NF\_NAT - self explanatory.

CONFIG\_IP\_NF\_TARGET\_LOG - self explanatory; glorious...

IP\_CONNTRACK\_FTP - we add this module for passive FTP.

IP\_NAT\_FTP - again, this module is for passive FTP.

IP\_CONNTRACK\_PROTOCOL\_DESTROY<sup>14</sup> - needed to make GRE work

IP\_CONNTRACK\_PROTOCOL\_UNREGISTER<sup>15</sup> - needed to make GRE

<sup>13</sup> Thanks to Oskar Andreasson's iptables tutorial 1.1.11, section 2.2, "Kernel Setup" at <http://www.netfilter.org/documentation/tutorials/blueflux/iptables-tutorial.html#AEN70>

<sup>14</sup> This module is currently found in the "Patch-O-Matic" system under "pending." Note that POM modules may break some other modules within Patch-O-Matic, and may have limitations. See <http://www.netfilter.org/documentation/FAQ/netfilter-faq-1.html#ss1.5> for information on Patch-O-Matic, [http://www.netfilter.org/documentation/pomlist/pom-submitted.html#ip\\_conntrack\\_protocol\\_destroy](http://www.netfilter.org/documentation/pomlist/pom-submitted.html#ip_conntrack_protocol_destroy) for details on the IP\_CONNTRACK\_PROTOCOL\_DESTROY module.

<sup>15</sup> This module is also found in the "Patch-O-Matic" system under "pending." See

work.

Getting the POM modules into the kernel involves running the POM command `./runme` pending and telling it where the kernel is located. After that comes configuring and compiling the custom kernel; we also have to build the iptables userland command. Don't forget to delete the older command, or you may get confused (and annoyed) when you accidentally use the wrong one and your scripts don't work!

Now that we have our machine up & running, we get to configure iptables. We could, in theory, do this line by line (as we were required to do for our Netopia R5300 router). However, this is both tedious and error-prone. Instead, we write a simple BASH script to do the job for us. This has several benefits: it's easier to proofread, it's easier to troubleshoot, and it's easier to reproduce (even on a different machine) if need be.

We'll include the entire script here, with comments explaining what's going on where necessary.

We start by defining simple variables. Even though they're all static values, doing so makes our script both easier to read, and easier to change in the future. For example, if we change ISP's and therefore have to change our public IP addresses, we only need to alter them in the variables up front, run the script, and we're right where we left off.

Note that with Netfilter, the order of rules is critical. This is because packets flow through sequentially, from first to last. We must filter out traffic that is invalid, such as spoofed source addresses or invalid TCP flag settings, before we accept anything based on destination. Performance is also affected by the order of the rules; we want to put the most common traffic types as close to the beginning as possible, in order to minimize the number of rules, and hence the amount of time, Netfilter has to scan before it can decide what to do with any given packet. So, we order first by security requirements, second by frequency of the packet type, and third for legibility.

Before we get to the actual script, let's take a look at the general logic flow of our firewall. We have flowcharts to help guide us through the maze of tables and chains; they don't necessarily show us the exact packet flow, but they do make it easier to visualize what's going on. We've added color backgrounds to the different chains in our script, to make references between the script and the network flowchart easier. These colors, of course, are not included in the plain-text script actually run on the firewall.

---

[http://www.netfilter.org/documentation/pomlist/pom-submitted.html#ip\\_conntrack\\_protocol\\_unregister](http://www.netfilter.org/documentation/pomlist/pom-submitted.html#ip_conntrack_protocol_unregister) for details.



# Firewall Flow

General Overview

This chart is based on the flowchart in Oskar Andreasson's iptables tutorial at "<http://www.netfilter.org/documentation/tutorials/blueflux/iptables-tutorial.html#TRAVERSINGOFTABLES>" -- thanks to Mr. Andreasson for the inspiration for this and the following flowcharts!

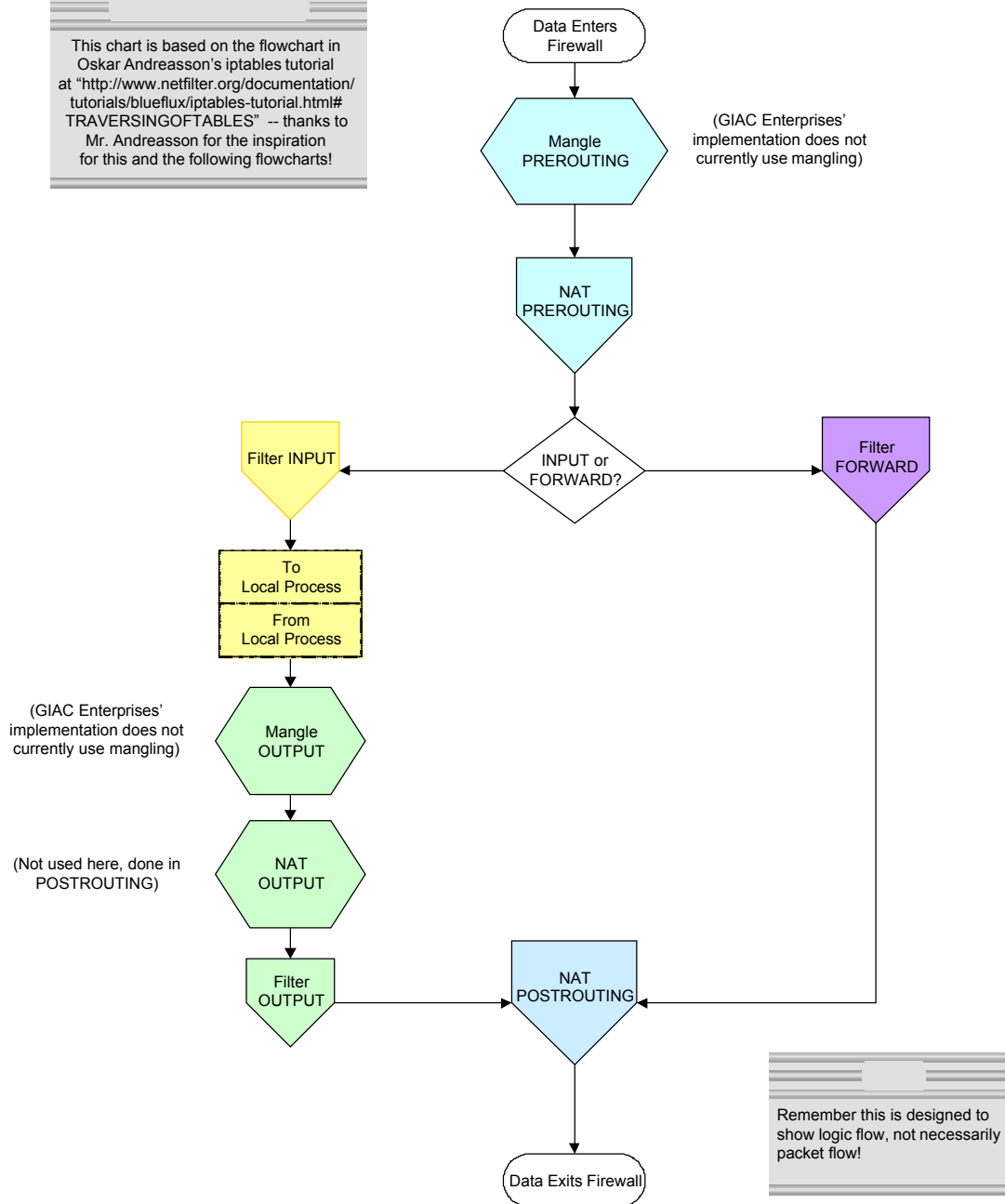


Figure 16, Netfilter's traffic flow

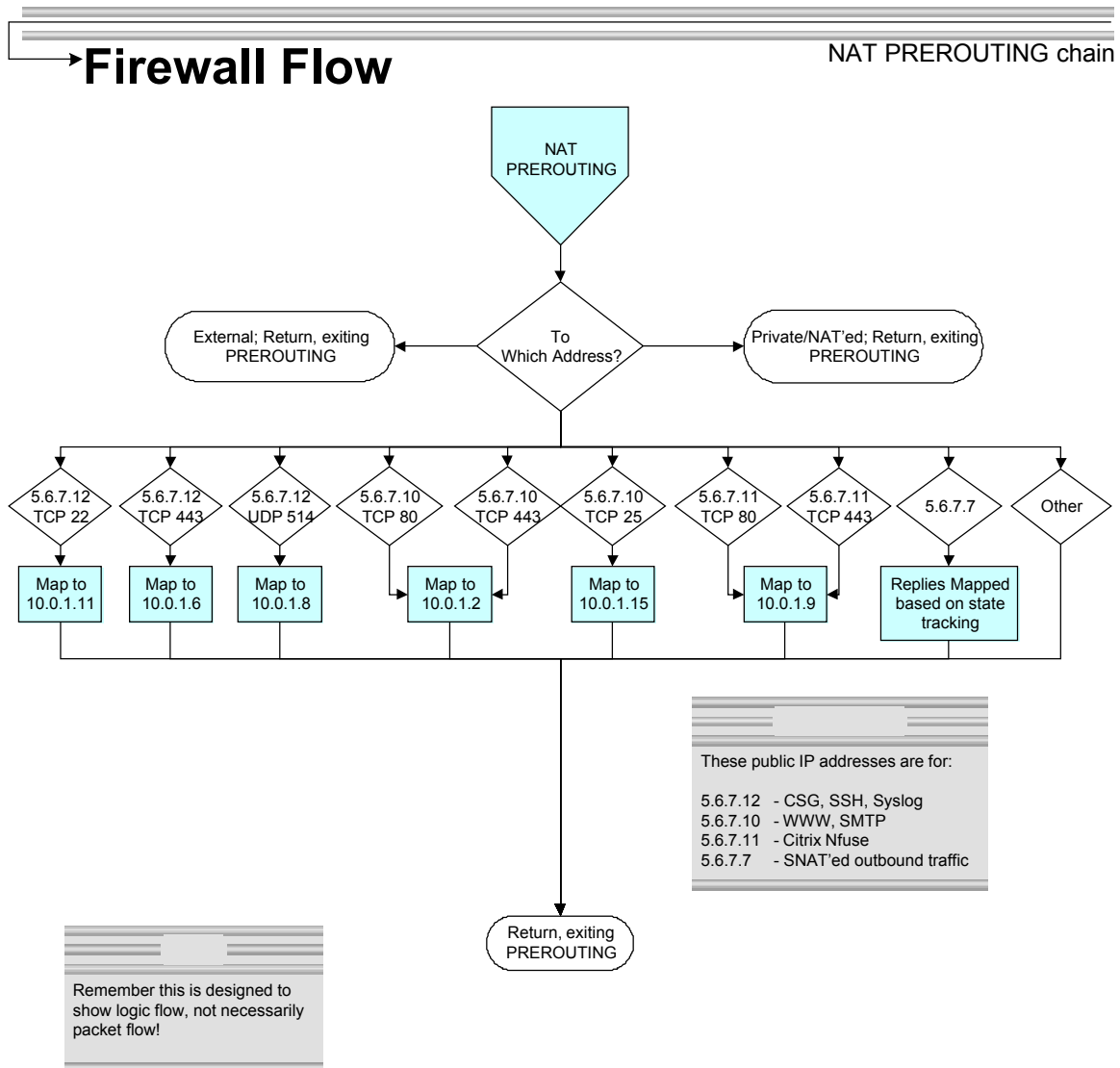


Figure 17, PREROUTING chain

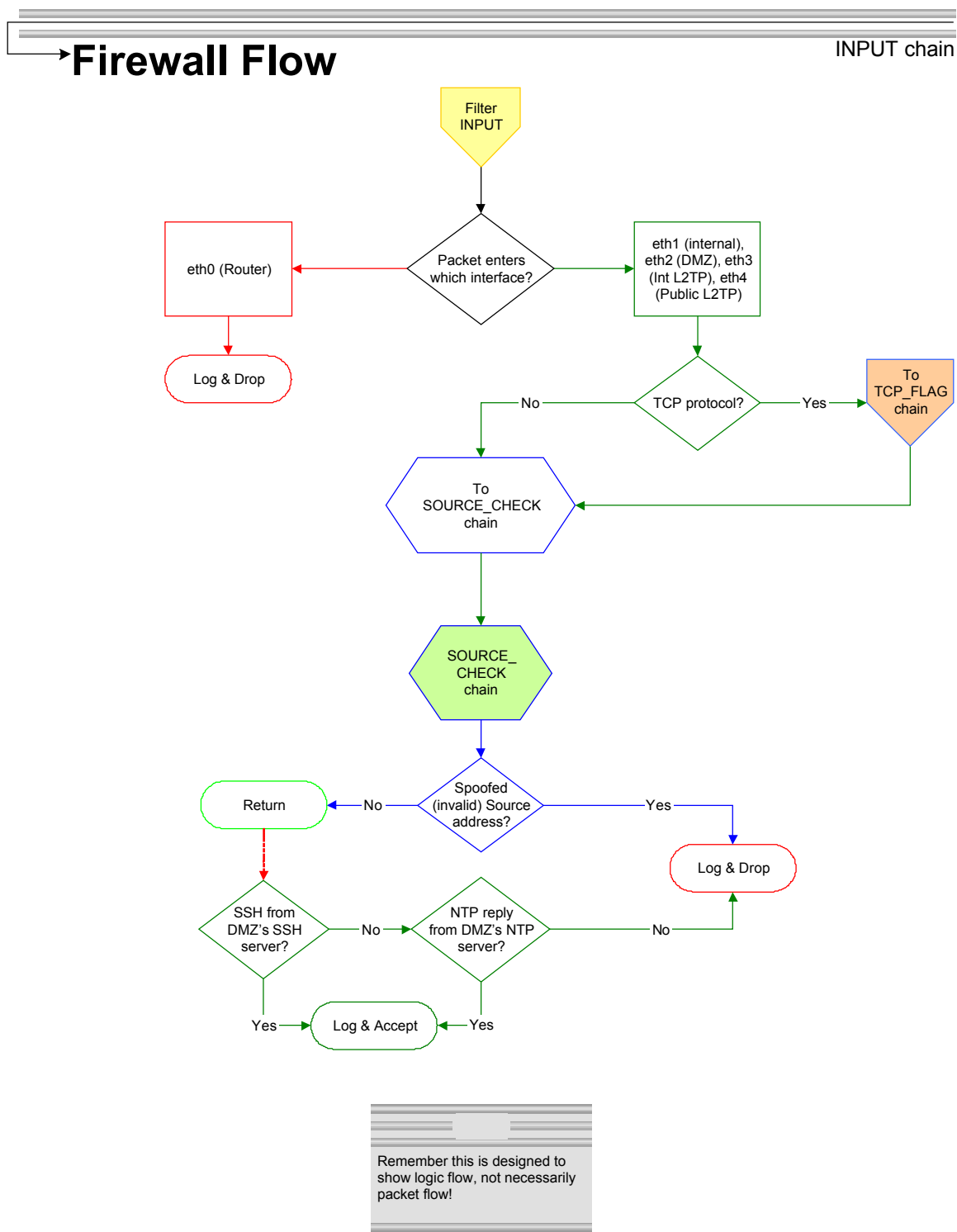


Figure 18, INPUT, TCP\_FLAG, and SOURCE\_CHECK chains



Figure 19, FORWARD, TCP\_FLAG, EX\_SOURCE\_CHECK chains

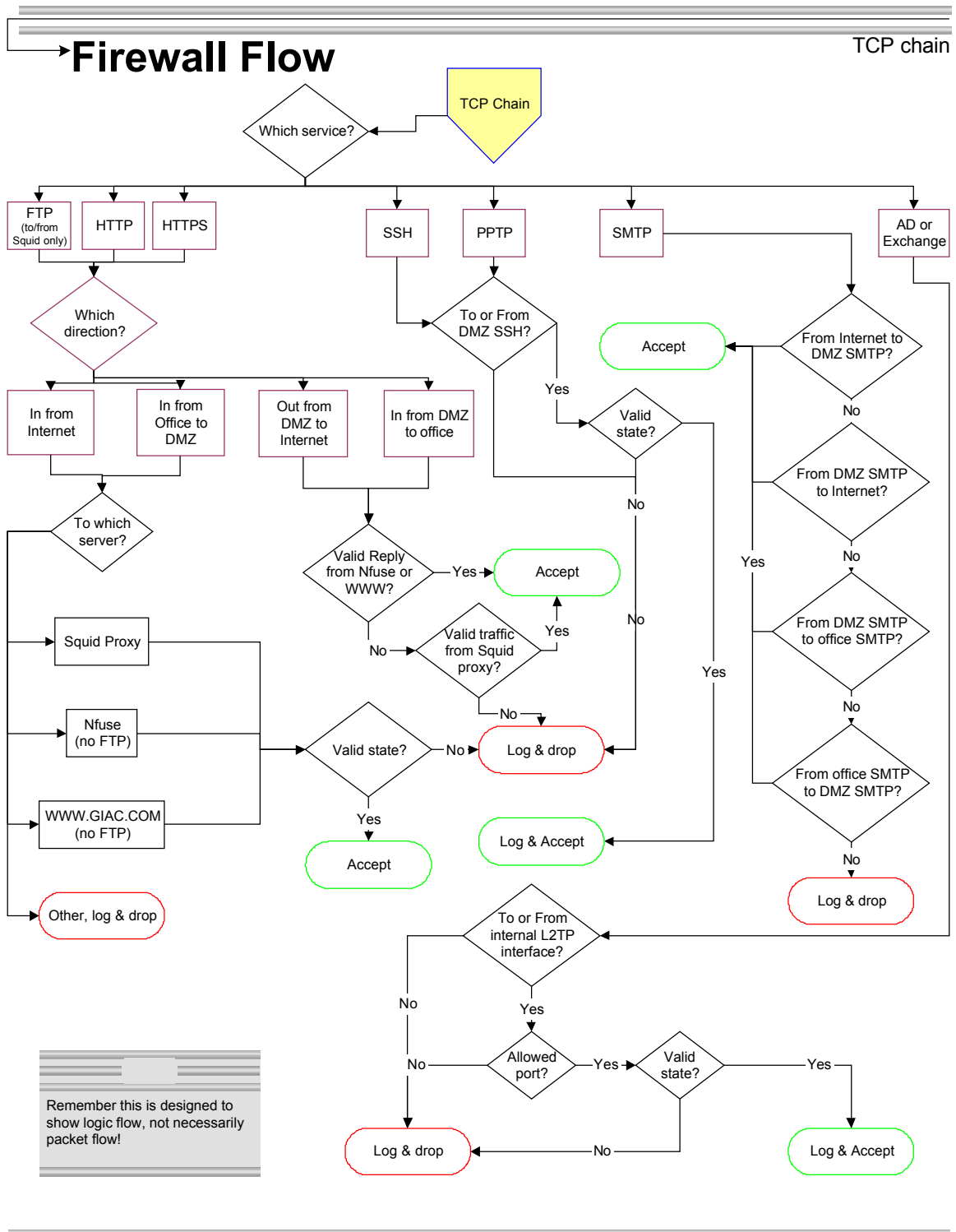
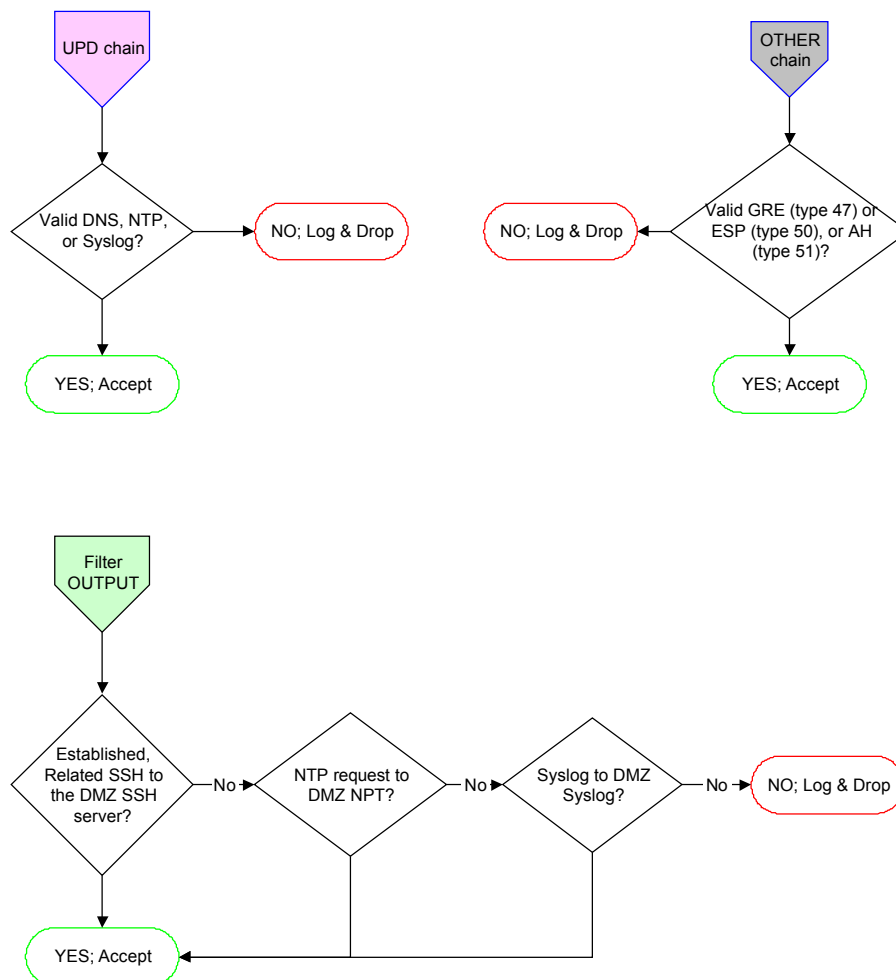


Figure 20, TCP chain

# Firewall Flow

UDP, OTHER, & OUTPUT chains



Remember this is designed to show logic flow, not necessarily packet flow!

Figure 21, UDP, OTHER, and OUTPUT chains

# Firewall Flow

ICMP & TCP\_FLAG chains

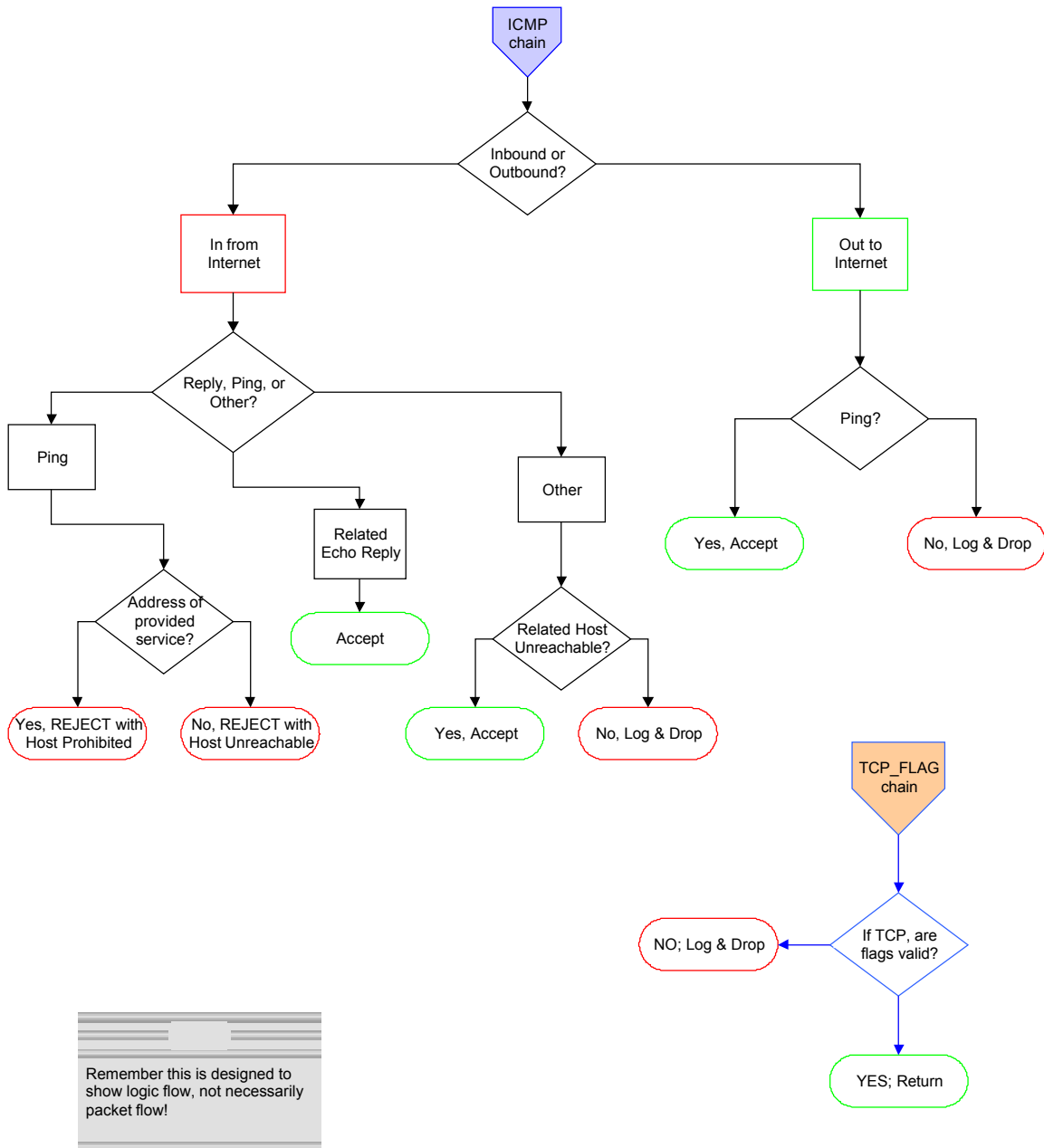


Figure 22, ICMP and TCP\_FLAG chains

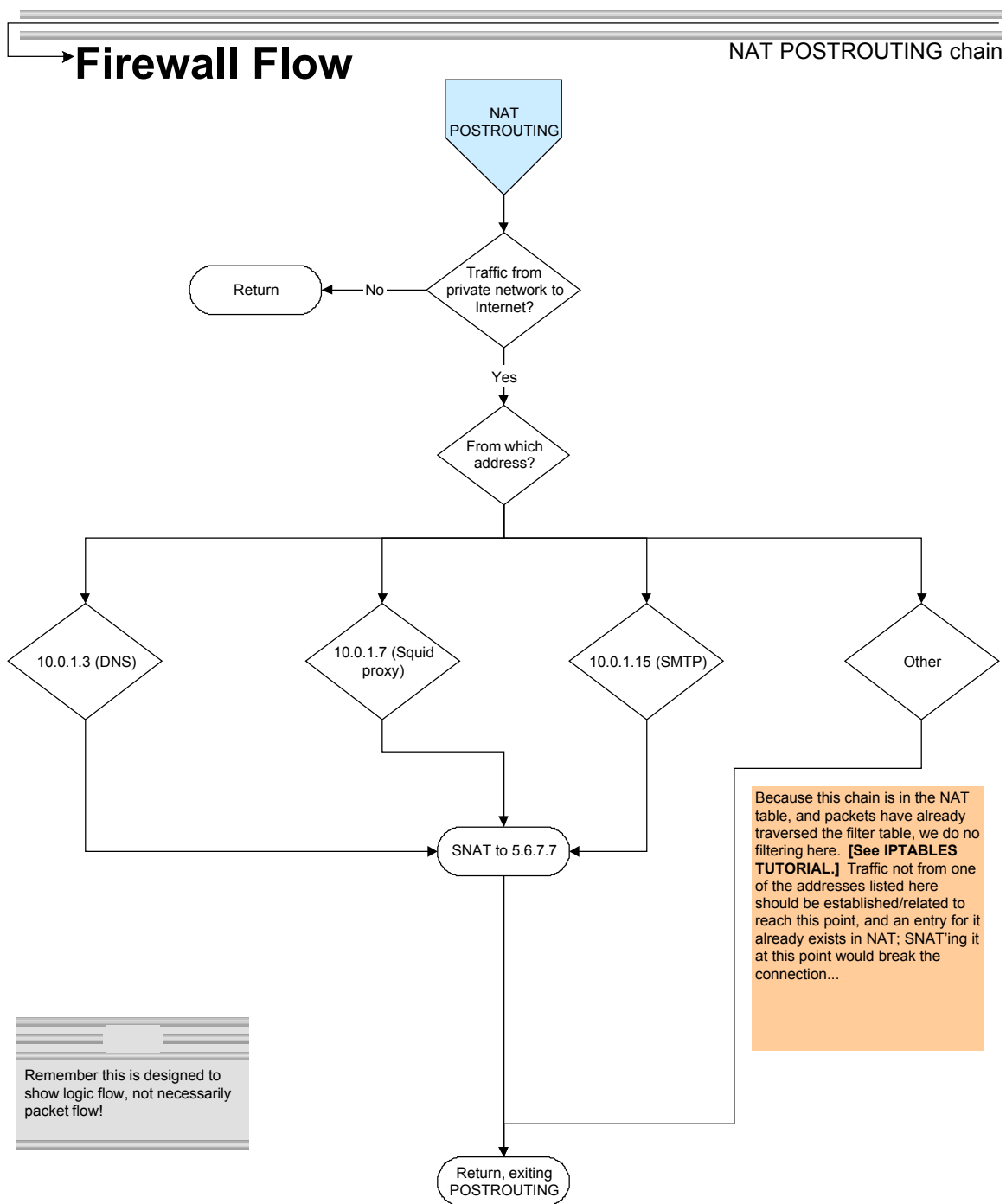


Figure 23, POSTROUTING chain



Now, let's take a look at our filter script.

```
#####  
#  
# GIAC Enterprises v.1.7 Netfilter Primary Firewall script  
#  
# Written by Vincent R. Streiff, Fall 2002  
#  
# This could not have been possible without guidance from the iptables  
# tutorial written by Oskar Andreason, located at:  
# http://www.netfilter.org/documentation/tutorials/blueflux/  
#  
# A plethora of other tutorials which may help can be found at:  
# http://www.netfilter.org/documentation/index.html#HOWTO  
#  
#  
# Our general guidelines:  
# Maximize performance, but not at the expense of security  
# Since complexity is bad, there may be instances where script  
# legibility takes precedence over streamlining for performance  
#  
#####  
  
#####  
#  
# For starters, to make all this legible and editable, we set our  
# variables used in the script  
#  
  
IPTABLES="/usr/local/sbin/iptables" # path to the iptables command; your  
# milage may vary depending on where you put it.  
EXT_IF="eth0" # This is the interface facing the Internet, connected to  
# our router  
INT_IF="eth1" # This is the interface facing the internal network  
DMZ_IF="eth2" # The interface to the service network (DMZ)  
VPN_EXT_IF="eth3" # Interface connected to the VPN's public interface  
VPN_INT_IF="eth4" # Interface connected to the VPN's internal interface  
PUB_NET="5.6.7.0/28" # This gives us 5.6.7.0-5.6.7.1.15  
INT_NET="10.0.0.0/23" # This gives us 10.0.0.0-10.0.1.255  
DMZ_NET="10.0.1.0/27" # This gives us 10.0.1.1-10.0.1.30 (usable) for DMZ  
L2TP_PRIV_NET="10.0.2.32/27" # Gives us 10.0.2.33-10.0.2.62 (usable) for L2TP  
EXT_ROUTER="5.6.7.1" # The address of our Netopia R5300 router  
PUB_NTP1="5.127.125.23" # Not really of course; you can often use ISP's  
# routers.  
PUB_NTP2="5.118.34.135" # We're using stratum 2 servers -- after notifying  
# the admins of those servers, of course -- to keep  
# ourselves as accurate as possible. We use 2 NTP  
# servers for redundancy; note that coordination with  
# the "official" time isn't as critical as keeping  
# all of our servers synchronized with each other.  
# See www.ntp.org for more on Network Time Protocol  
PUB_FW="5.6.7.9" # Public address of our firewall (eth0)  
DMZ_FW="10.0.1.1" # IP address of our firewall's DMZ NIC (eth2)  
INT_FW="10.0.0.1" # IP address of firewall's internal NIC (eth1)  
PUB_FW_L2TP="5.6.7.2" # IP address of firewall's NIC connected to PUB_L2TP  
DMZ_FW_L2TP="10.0.2.1" # IP address of firewalls NIC connected to DMZ_L2TP  
PUB_L2TP="5.6.7.8" # External, public IP address of our L2TP VPN  
DMZ_L2TP="10.0.2.12" # IP address of our L2TP inward facing NIC
```

---

```

PUB_WWW="5.6.7.10" # Public IP address of the GIAC.COM Web server
DMZ_WWW="10.0.1.2" # Private, "real" IP address of the GIAC.COM
# Web server
PUB_NFUSE="5.6.7.11" # Public IP address of Citrix NFuse Web Server
DMZ_NFUSE="10.0.1.9" # Private, "real" IP address of NFuse server
PUB_CSG="5.6.7.12" # Public IP address for Citrix Secure Gateway
DMZ_CSG="10.0.1.6" # Private, "real" IP address for Citrix Secure
# Gateway; note that this is the same server as
# NFuse, but uses a different NIC and addresses
# since we need inbound TCP 443 to both CSG and NFuse
# See http://hqextsrsvft01.citrix.com/cgi-
# bin/webcgi.exe/?Session=4822373,U=1,ST=187,N=0005,
# K=1249,SXI=12,Case=obj(13815) for how to get Citrix
# Secure Gateway and NFuse to run on the same server. (Note:
# URL wraps across multiple lines)
# Also see MS TechNet article Q238131, How to
# disable socket pooling
PUB_SSH="5.6.7.12" # Public IP address of our SSH server
DMZ_SSH="10.0.1.11" # Private, "real" IP address of our SSH server
PUB_SMTP="5.6.7.10" # Public IP address of our mail server
DMZ_SMTP="10.0.1.15" # Private, "real" IP address for mail relay
INT_SMTP="10.0.0.15" # IP address of our Exchange server
INT_VIP="10.0.0.0/27" # 10.0.0.1-10.0.0.30 to use for servers
INT_IT_NET="10.0.0.32/27" # Addresses for IT staff's PC's
# (10.0.0.33-10.0.0.62)
INT_CITRIX1="10.0.0.29" # Citrix Metaframe server number 1
INT_CITRIX2="10.0.0.30" # Citrix Metaframe server number 2
INT_NET_CLIENTS="10.0.0.128/26" # Our staff PC's, assigned via DHCP; this
# gives us 10.0.0.129-10.0.0.190 to assign to PCs
# Note that if the office expands, we've still
# got plenty of unused addresses; if needed, we
# can always use additional addresses within
# the 10.0.0.0/24 subnet
PUB_SYSLOG="5.6.7.12" # This is so our router can send syslog info
INT_SYSLOG="10.0.0.8" # IP address of our internal Syslog server
DMZ_SYSLOG="10.0.1.8" # IP address of our syslog server on the DMZ
DMZ_SQUID="10.0.1.7" # IP address of our Squid proxy server
DMZ_DNS="10.0.1.3" # IP address of "caching" DNS server in the DMZ
INT_DNS="10.0.0.21" # IP address of our internal DNS server (we
# have 2, but since Windows clients generally
# won't use the secondary even if configured, we just deny
# the backup DNS access - so no reference needed.
DMZ_NTP="10.0.1.8" # IP address of our DMZ's time server
INT_NTP="10.0.0.8" # IP address of internal network time server
DMZ_SQL="10.0.1.28" # IP address of the MS SQL 2000 server
INT_DC1="10.0.0.21" # IP address of the AD Global Catalogue DC
INT_DC2="10.0.0.22" # IP address of second AD domain controller
INT_DC3="10.0.0.23" # IP address of third AD domain controller
DMZ_SUS="10.0.1.17" # IP address of DMZ Windows Update & Norton A/V Server
INT_SUS="10.0.0.17" # IP address of internal Windows Update Server
INT_NAV="10.0.0.22" # Address of the internal Norton Antivirus server
DMZ_NAV="10.0.1.17" # Address of the DMZ Norton Antivirus server
SNAT_IP="5.6.7.7" # IP address used for NATing our public traffic
UNUSED_IP1="5.6.7.3" # Unused IP address in our public subnet
UNUSED_IP2="5.6.7.4" # Unused IP address in our public subnet
UNUSED_IP3="5.6.7.5" # Unused IP address in our public subnet
UNUSED_IP4="5.6.7.6" # Unused IP address in our public subnet

```

```

####
#
# Make sure Netfilter is pristine before we start
#

## First, we flush things out, just to be safe
#
$IPTABLES -F
$IPTABLES -X          # Delete any existing custom chains before we start
$IPTABLES -t nat -F
$IPTABLES -t mangle -F # We aren't using this table, but still want it clean

##
# Next, we set the default policies for our tables; we drop by default
# according to our policy of least required privilege; we only want traffic
# we specifically allow to get through.
$IPTABLES -P INPUT DROP      # Default policy of Drop
$IPTABLES -P FORWARD DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -t nat -P PREROUTING ACCEPT # We want to avoid filtering here16
$IPTABLES -t nat -P POSTROUTING ACCEPT # No filtering here, that's been done
$IPTABLES -t nat -P OUTPUT ACCEPT
$IPTABLES -t mangle -P PREROUTING ACCEPT # We avoid filtering here
$IPTABLES -t mangle -P INPUT ACCEPT
$IPTABLES -t mangle -P FORWARD ACCEPT
$IPTABLES -t mangle -P OUTPUT ACCEPT
$IPTABLES -t mangle -P POSTROUTING ACCEPT

####
#
# Kernel preparations:
# Most scripts have them at the beginning, we're putting them at the
# end to avoid the momentary exposure -- Netfilter lets us set rules
# before interfaces even exist, so we can wait until it's ready before
# we turn on any routing
#
# This would also be a good place to add any modules that weren't compiled in
# the kernel. We don't need to do that, though.

##
# Now we get to the real fun, writing filters!

# First, we create our "custom" chains

# For the filter table -- i.e. INPUT, FORWARD, & OUTPUT
$IPTABLES -t filter -N TCP_FLAG
$IPTABLES -t filter -N SOURCE_CHECK
$IPTABLES -t filter -N EX_SOURCE_CHECK
$IPTABLES -t filter -N ICMP
$IPTABLES -t filter -N TCP
$IPTABLES -t filter -N UDP
$IPTABLES -t filter -N OTHER

```

---

<sup>16</sup> See <http://www.netfilter.org/documentation/tutorials/blueflux/iptables-tutorial.html#TABLES> which states in part, "The rest of the packets in the stream will... not go through this table again, but instead they will automatically have the same actions taken to them as the first packet in the stream. This is one reason why you should not do any filtering in this table..."

```
####
#
# Start with the PREROUTING chain, so we've got a logical, legible flow here
#
$IPTABLES -t nat -A PREROUTING -d $PUB_WWW -p tcp --dport 80 -j DNAT \
--to-destination $DMZ_WWW
$IPTABLES -t nat -A PREROUTING -d $PUB_WWW -p tcp --dport 443 -j DNAT \
--to-destination $DMZ_WWW
$IPTABLES -t nat -A PREROUTING -d $PUB_NFUSE -p tcp --dport 80 -j DNAT \
--to-destination $DMZ_NFUSE
$IPTABLES -t nat -A PREROUTING -d $PUB_NFUSE -p tcp --dport 443 -j DNAT \
--to-destination $DMZ_NFUSE
$IPTABLES -t nat -A PREROUTING -d $PUB_CSG -p tcp --dport 443 -j DNAT \
--to-destination $DMZ_CSG
$IPTABLES -t nat -A PREROUTING -d $PUB_SSH -p tcp --dport 22 -j DNAT \
--to-destination $DMZ_SSH
$IPTABLES -t nat -A PREROUTING -d $PUB_SMTP -p tcp --dport 25 -j DNAT \
--to-destination $DMZ_SMTP
$IPTABLES -t nat -A PREROUTING -d $PUB_SYSLOG -p udp --dport 514 -j DNAT \
--to-destination $DMZ_SYSLOG
```

```
####
#
# INPUT chain - any traffic going TO the firewall itself
#
# First, check for invalid flags & invalid sources
# We check for invalid flags first for logging purposes -- assume
# most would also be caught with source_check, but
# want to be able to find them in the logs!
#
$IPTABLES -A INPUT -p tcp -j TCP_FLAG
$IPTABLES -A INPUT -j SOURCE_CHECK
$IPTABLES -A INPUT -i $EXT_IF -j EX_SOURCE_CHECK

# Note on EX_SOURCE_CHECK: all traffic coming into the EXT_IF interface will
# get dropped anyway, but we still want to check it for logging purposes.
# Note also that if we're seeing invalid sources here, it means our border
# router's filtering has failed!!

# We don't actually allow ANY direct access to the firewall from the Internet
# or the internal network, so if it hasn't been dropped & logged already,
# do so now. Also implement rate limiting, to help mitigate DoS attacks.
#
$IPTABLES -A INPUT -i $EXT_IF -m limit --limit 3 -j LOG --log-level warn \
--log-prefix "FW ACCESS ATTEMPT: "
$IPTABLES -A INPUT -i $EXT_IF -j DROP
$IPTABLES -A INPUT -i $INT_IF -m limit --limit 3 -j LOG --log-level warn \
--log-prefix "FW ACCESS ATTEMPT: "
$IPTABLES -A INPUT -i $INT_IF -j DROP

# Now we analyze -- it would be easy to just filter everything we want
# right here, since the only traffic we allow to the firewall itself is SSH
# from DMZ & NTP replies. It'll be easier to cross-check open ports later,
# though, if we check it in the TCP & UDP chains. Note also that doing it
```

```
# here instead would be more efficient; we don't expect a performance
# problem, though, so this shouldn't be an issue.
```

```
$IPTABLES -A INPUT -p tcp -j TCP
$IPTABLES -A INPUT -p udp -j UDP
```

```
# Log & drop anything else that made it this far through the INPUT chain
#
```

```
$IPTABLES -A INPUT -j LOG --log-level warn --log-prefix "FW ACCESS ATTEMPT: "
$IPTABLES -A INPUT -j DROP
```

```
####
```

```
#
# FORWARD chain - filters anything going THROUGH the firewall
#
```

```
# First, check for invalid flags & invalid sources.
# We check for invalid flags first for logging purposes -- assume
# most would probably be caught with source_check, but
# want to be able to find them in the logs!
#
```

```
$IPTABLES -A FORWARD -p tcp -j TCP_FLAG
$IPTABLES -A FORWARD -j SOURCE_CHECK
$IPTABLES -A FORWARD -i $EXT_IF -j EX_SOURCE_CHECK
# Note that if we're seeing invalid sources here, it means our border router's
# filtering has failed!!
```

```
#
# If the traffic made it through, we check to see if we want it. We break it
# down by protocol so it's easier to read; plus, we can list rules by protocol
# to ease cross-checking later, after we're up & running
#
```

```
# Order is important here. We jump to TCP first, since that's the bulk
# of our traffic; the OTHER chain goes at the end because we can't
# exclude a range or group of protocols; by putting it at the end, traffic for
# tcp, udp, and icmp has already been filtered out, leaving only the other
# protocols such as GRE and ESP. If, for some reason, traffic comes back here
# after jumping to one of the first three chains, it will go through the OTHER
# chain, but that really doesn't hurt anything; it's just one more place to
# drop it...
```

```
$IPTABLES -A FORWARD -p tcp -j TCP
$IPTABLES -A FORWARD -p udp -j UDP
$IPTABLES -A FORWARD -p icmp -j ICMP
$IPTABLES -A FORWARD -j OTHER
```

```
# If it made it back here somehow, log for analysis & drop
#
```

```
$IPTABLES -A FORWARD -j LOG --log-level warn --log-prefix "FUNKY TRAFFIC: "
$IPTABLES -A FORWARD -j DROP
```

```
####
```

```
# TCP_FLAG Chain
#
```

```

# Checks for invalid TCP Flags
#
#

$IPTABLES -A TCP_FLAG -p tcp ! --syn -m state --state NEW -j LOG \
    --log-level debug --log-prefix "NEW not SYN: "
$IPTABLES -A TCP_FLAG -p tcp ! --syn -m state --state NEW -j DROP
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags ALL NONE -j LOG --log-level warn \
    --log-prefix "TCP FLAGS = NONE: "
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags ALL NONE -j DROP

#
# We tell iptables which flags to examine, followed by which flags to match
#
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags ALL ALL -j LOG --log-level warn \
    --log-prefix "TCP FLAGS = ALL: "
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags ALL ALL -j DROP
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags FIN,PSH,URG FIN,PSH,URG -j LOG \
    --log-level warn --log-prefix "TCP FLAGS = CHRISTMAS: "
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags FIN,PSH,URG FIN,PSH,URG -j DROP
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags FIN,SYN FIN,SYN -j LOG \
    --log-level warn --log-prefix "TCP FLAGS = SYN-FIN: "
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags FIN,SYN FIN,SYN -j DROP
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags SYN,RST SYN,RST -j LOG \
    --log-level warn --log-prefix "TCP FLAGS = SYN-RST: "
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags SYN,RST SYN,RST -j DROP
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags FIN,RST FIN,RST -j LOG \
    --log-level warn --log-prefix "TCP FLAGS = FIN-RST: "
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags FIN,RST FIN,RST -j DROP
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags FIN,ACK FIN -j LOG --log-level warn \
    --log-prefix "TCP FLAGS = FIN no ACK: "
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags FIN,ACK FIN -j DROP
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags PSH,ACK PSH -j LOG --log-level warn \
    --log-prefix "TCP FLAGS = PSH no ACK: "
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags PSH,ACK PSH -j DROP
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags URG,ACK URG -j LOG --log-level warn \
    --log-prefix "TCP FLAGS = URG no ACK: "
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags URG,ACK URG -j DROP

```

```

####
#
# SOURCE_CHECK chain
#
# Checks for invalid (reserved/spoofed) source address
#
# Note: this could be streamlined somewhat with better masking in some areas
# (e.g. /3 instead of /8); that might be somewhat more efficient, but for now
# we're listing each Class A address block individually to make the script
# easier to read. We could also use some fancier scripting, rather than doing
# this line by line. Again, this method is simple to read and understand.
#
# See http://www.iana.org/assignments/ipv4-address-space for up-to-date
# listing of assigned addresses.

$IPTABLES -A SOURCE_CHECK -s 1.0.0.0/8 -j LOG --log-level warn \
    --log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 1.0.0.0/8 -j DROP

```

```

$IPTABLES -A SOURCE_CHECK -s 2.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 2.0.0.0/8 -j DROP

##
#
# NOTE: In reality, we also include the 5.0.0.0/8 block; for purposes of this
# text, however, we comment it out since this is the address we're pretending
# to use. Note that in reality, this is of course not the address block GIAC
# Enterprises uses!
#
### $IPTABLES -A SOURCE_CHECK -s 5.0.0.0/8 -j LOG --log-level warn \
### --log-prefix "SPOOFSOURCE: "
### $IPTABLES -A SOURCE_CHECK -s 5.0.0.0/8 -j DROP
#

$IPTABLES -A SOURCE_CHECK -s 7.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 7.0.0.0/8 -j DROP

#
# We leave this next address block for the "EX_SOURCE_CHECK" so we don't drop
# all of our own traffic!
#
### $IPTABLES -A SOURCE_CHECK -s 10.0.0.0/8 -j LOG --log-level warn \
### --log-prefix "SPOOFSOURCE: "
### $IPTABLES -A SOURCE_CHECK -s 10.0.0.0/8 -j DROP
#

$IPTABLES -A SOURCE_CHECK -s 23.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 23.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 27.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 27.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 31.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 31.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 36.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 36.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 37.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 37.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 39.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 39.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 41.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 41.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 42.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 42.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 49.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 49.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 50.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "

```

```

$IPTABLES -A SOURCE_CHECK -s 50.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 58.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 58.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 59.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 59.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 60.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 60.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 69.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 69.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 70.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 70.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 71.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 71.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 72.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 72.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 73.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 73.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 74.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 74.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 75.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 75.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 76.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 76.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 77.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 77.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 78.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 78.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 79.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 79.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 82.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 82.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 83.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 83.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 84.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 84.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 85.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 85.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 86.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 86.0.0.0/8 -j DROP

```



```

$IPTABLES -A SOURCE_CHECK -s 87.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 87.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 88.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 88.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 89.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 89.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 90.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 90.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 91.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 91.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 92.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 92.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 93.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 93.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 94.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 94.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 95.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 95.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 96.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 96.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 97.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 97.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 98.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 98.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 99.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 99.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 100.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 100.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 101.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 101.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 102.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 102.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 103.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 103.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 104.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 104.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 105.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 105.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 106.0.0.0/8 -j LOG --log-level warn \

```

```

--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 106.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 107.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 107.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 108.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 108.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 109.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 109.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 110.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 110.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 111.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 111.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 112.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 112.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 113.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 113.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 114.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 114.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 115.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 115.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 116.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 116.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 117.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 117.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 118.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 118.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 119.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 119.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 120.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 120.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 121.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 121.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 122.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 122.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 123.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 123.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 124.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 124.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 125.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "

```

```

$IPTABLES -A SOURCE_CHECK -s 125.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 126.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 126.0.0.0/8 -j DROP

#
# Don't log or drop legitimate loopback traffic!
#
$IPTABLES -A SOURCE_CHECK -s 127.0.0.0/8 -i ! lo -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 127.0.0.0/8 -i ! lo -j DROP

$IPTABLES -A SOURCE_CHECK -s 197.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 197.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 221.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 221.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 222.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 222.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 223.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 223.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 240.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 240.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 241.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 241.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 242.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 242.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 243.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 243.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 244.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 244.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 245.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 245.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 246.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 246.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 247.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 247.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 248.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 248.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 249.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 249.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 250.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 250.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 251.0.0.0/8 -j LOG --log-level warn \

```

```

--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 251.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 252.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 252.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 253.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 253.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 254.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 254.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 255.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 255.0.0.0/8 -j DROP

```

```

####
#
# EX_SOURCE_CHECK chain
#
# Check to make sure packets from the outside aren't spoofing our internal
# address
#
$IPTABLES -A EX_SOURCE_CHECK -i $EXT_IF -s 10.0.0.0/8 -j LOG --log-level \
warn --log-prefix "SPOOFSOURCE: "
$IPTABLES -A EX_SOURCE_CHECK -i $EXT_IF -s 10.0.0.0/8 -j DROP

```

```

####
#
# TCP chain - filters all tcp traffic (IP protocol 6)
#
# Allow only the traffic we really need, obviously, based on our Security
# Policy; that breaks down to SSH,HTTP,HTTPS/SSL,SMTP,Passive FTP, L2TP.
# We've tried to put the most frequently used traffic first; note that when
# we use the iptables -L <CHAIN> -v -n --line-numbers command later to analyze
# and audit our rulesets, we'll be able to see the amount of traffic matching
# each rule. We will then be able to adjust order if/as needed to improve
# performance. In the meantime, we're making an educated guess on how to
# start. We expect most of our traffic to be outbound Web traffic from
# internal staff.
# We're using a Squid proxy; we only allow this traffic from the clients (not
# our servers) to the proxy, and from the proxy to the Internet. Replies
# follow the reverse path. The connection tracking will enable us to ensure
# replies are valid.
#
# Lastly, note that we'll be using Netfilter on the Squid proxy itself;
# if we need to do any port redirecting/forwarding for Squid itself, we'll do
# it there. This way, we have a "transparent" proxy for our clients; our
# primary firewall's script meanings are obvious; and we can reconfigure
# Squid if desired without have to touch our firewall's settings.
#
# Office Web traffic from inside to the DMZ proxy:
$IPTABLES -A TCP -i $INT_IF -o $DMZ_IF -s $INT_NET_CLIENTS -d $DMZ_SQUID \
-p tcp --dport 80 -m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $DMZ_IF -s $INT_NET_CLIENTS -d $DMZ_SQUID \
-p tcp --dport 443 -m state --state NEW -j ACCEPT

# Office Web traffic from the proxy to the Internet:
$IPTABLES -A TCP -i $DMZ_IF -o $EXT_IF -s $DMZ_SQUID -p tcp --dport 80 \

```

```

-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $DMZ_IF -o $EXT_IF -s $DMZ_SQUID -p tcp --dport 443 \
-m state --state NEW -j ACCEPT

# Office Web traffic from the Internet back to the proxy (valid replies only!)
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -d $DMZ_SQUID -p tcp --sport 80 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -d $DMZ_SQUID -p tcp --sport 443 \
-m state --state ESTABLISHED,RELATED -j ACCEPT

# Office Web traffic from the proxy back to the inside (valid replies only!)
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_SQUID -d $INT_NET_CLIENTS \
-p tcp --sport 80 -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_SQUID -d $INT_NET_CLIENTS \
-p tcp --sport 443 -m state --state ESTABLISHED,RELATED -j ACCEPT

# Next on our list comes e-mail (SMTP). We allow both inbound and outbound;
# like the Web traffic above, this goes through the DMZ in both directions.
# We use our Postfix SMTP relay both to protect our Exchange server from
# direct exposure to the Internet, and to filter out the headers as well to
# disguise from the rest of the world doesn't even know we using Exchange.
# Mail's not worth logging, too much traffic!

# Inbound mail, from Internet to DMZ (Postfix SMTP relay)
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -d $DMZ_SMTP -p tcp --dport 25 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $DMZ_IF -o $EXT_IF -s $DMZ_SMTP -p tcp --sport 25 \
-m state --state ESTABLISHED,RELATED -j ACCEPT

# Inbound mail from DMZ relay to internal Exchange
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_SMTP -d $INT_SMTP -p tcp \
--dport 25 -m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $DMZ_IF -s $INT_SMTP -d $DMZ_SMTP -p tcp \
--dport 25 -m state --state ESTABLISHED,RELATED -j ACCEPT

# Outbound mail traffic, from Exchange to Postfix
$IPTABLES -A TCP -i $INT_IF -o $DMZ_IF -s $INT_SMTP -d $DMZ_SMTP -p tcp \
--dport 25 -m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_SMTP -d $INT_SMTP -p tcp \
--sport 25 -m state --state ESTABLISHED,RELATED -j ACCEPT

# Outbound mail from Postfix relay to the Internet
$IPTABLES -A TCP -i $DMZ_IF -o $EXT_IF -s $DMZ_SMTP -p tcp --dport 25 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -d $DMZ_SMTP -p tcp --sport 25 \
-m state --state ESTABLISHED,RELATED -j ACCEPT

# Next, we need to let the world reach our public www.giac.com Web server.
# Too much to log here, so we'll have to rely on the logs from our IDS and the
# Web server itself. We'll add in some rate limits here, since it's a likely
# DoS target.

# We allow HTTP, HTTPS from the Internet to our Web server
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -d $DMZ_WWW -p tcp --dport 80 \
-m state --state NEW -m limit --limit 3 -j ACCEPT
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -d $DMZ_WWW -p tcp --dport 443 \

```

```

-m state --state NEW -m limit --limit 3 -j ACCEPT
$IPTABLES -A TCP -i $DMZ_IF -o $EXT_IF -s $DMZ_WWW -p tcp --sport 80 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $DMZ_IF -o $EXT_IF -s $DMZ_WWW -p tcp --sport 443 \
-m state --state ESTABLISHED,RELATED -j ACCEPT

# We also allow our internal staff to get to the server.
$IPTABLES -A TCP -i $INT_IF -o $DMZ_IF -s $INT_NET_CLIENTS -d $DMZ_WWW \
-p tcp --sport 80 -m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $DMZ_IF -s $INT_NET_CLIENTS -d $DMZ_WWW \
-p tcp --sport 443 -m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_WWW -d $INT_NET_CLIENTS \
-p tcp --sport 80 -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_WWW -d $INT_NET_CLIENTS \
-p tcp --sport 443 -m state --state ESTABLISHED,RELATED -j ACCEPT

# Next, we allow external staff to access the Citrix NFuse server. This also
# requires access to the Citrix Secure Gateway. (Note that without CSG, we
# would have to open up port 1494 from the Internet to our internal Citrix
# Metaframe servers!)
# We want to log this. We allow both HTTP and HTTPS, though the HTTP access
# just results in a message telling the client to use HTTPS.

$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -d $DMZ_NFUSE -p tcp --dport 80 \
-m state --state NEW -j LOG --log-level debug --log-prefix \
"NFUSE ACCESS: "
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -d $DMZ_NFUSE -p tcp --dport 80 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -d $DMZ_NFUSE -p tcp --dport 443 -m \
state --state NEW -j LOG --log-level debug --log-prefix "NFUSE ACCESS: "
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -d $DMZ_NFUSE -p tcp --dport 443 -m \
state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -d $DMZ_CSG -p tcp --dport 443 \
-m state --state NEW -j LOG --log-level debug --log-prefix "CSG ACCESS: "
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -d $DMZ_CSG -p tcp --dport 443 \
-m state --state NEW -j ACCEPT

# And, of course, we need the replies to this NFuse & CSG traffic
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_NFUSE -p tcp --sport 80 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_NFUSE -p tcp --sport 443 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_CSG -p tcp --sport 443 \
-m state --state ESTABLISHED,RELATED -j ACCEPT

# Citrix NFuse will use the Citrix SSL Relay to authenticate users on the
# MetaFrame servers, so we need 443 open to them as well.

$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_NFUSE -d $INT_CITRIX1 -p tcp \
--dport 443 -m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_NFUSE -d $INT_CITRIX2 -p tcp \
--dport 443 -m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $DMZ_IF -s $INT_CITRIX1 -d $DMZ_NFUSE -p tcp \
--sport 443 -m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $DMZ_IF -s $INT_CITRIX2 -d $DMZ_NFUSE -p tcp \
--sport 443 -m state --state NEW -j ACCEPT

# CSG also requires access to the Citrix Metaframe servers. Note that this

```

```
# traffic will also be using IPSec; IPSec's needs are handled in the UDP and
# OTHER chains.
```

```
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_CSG -d $INT_CITRIX1 -p tcp \
--dport 1494 -m state --state NEW -j LOG --log-level debug --log-prefix \
"ICA: "
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_CSG -d $INT_CITRIX1 -p tcp \
--dport 1494 -m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_CSG -d $INT_CITRIX2 -p tcp \
--dport 1494 -m state --state NEW -j LOG --log-level debug --log-prefix \
"ICA: "
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_CSG -d $INT_CITRIX2 -p tcp \
--dport 1494 -m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $DMZ_IF -s $INT_CITRIX1 -d $DMZ_CSG -p tcp \
--sport 1494 -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $DMZ_IF -s $INT_CITRIX2 -d $DMZ_CSG -p tcp \
--sport 1494 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

```
# We need to allow our internal Microsoft Software Update Server to get to the
# SUS server on the DMZ.
```

```
$IPTABLES -A TCP -i $INT_IF -o $DMZ_IF -s $INT_SUS -d $DMZ_SUS -p tcp \
--dport 80 -m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_SUS -d $INT_SUS -p tcp \
--sport 80 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

```
# Similarly, we need to allow Norton Antivirus to update
```

```
$IPTABLES -A TCP -i $INT_IF -o $DMZ_IF -s $INT_NAV -d $DMZ_NAV -p tcp \
--dport 21 -m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_NAV -d $INT_NAV -p tcp \
--sport 21 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

```
# The other VPN we're providing for staff is L2TP. That runs over UDP -- we
# still need to open up a whole lot of stuff for the internal L2TP interface
# connection to and from the internal network. We'll put the TCP parts of
# that here.
```

```
# Traffic to the Global Catalogue servers (and replies as well):
```

```
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC1 -p tcp --dport 3268 \
-m state --state NEW -j LOG --log-level debug --log-prefix "GC ACCESS: "
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC1 -p tcp --dport 3268 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC1 -p tcp --sport 3268 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC2 -p tcp --dport 3268 \
-m state --state NEW -j LOG --log-level debug --log-prefix "GC ACCESS: "
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC2 -p tcp --dport 3268 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC2 -p tcp --sport 3268 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
```

```
# Other traffic to our Domain Controllers. We have 3, including the GC;
# our tests failed unless the VPN server has access to all. There may be a
# registry setting somewhere to fix that, but until we find it we'll have to
# open up holes to all three. Note that because of the security implications
# of opening up these ports, many organizations simplify their rules by just
# allowing all communications, rather than specifying all of these.
```



```

# Our belief (and our policy) is that if there's no need for access, there
# should be no access.
#
# What we have here are Kerberos, RPC, Netlogon, RPC Service ports, LDAP,
# Netbios and of course all the replies associated with this traffic.
# These ports are further explained in the following text on the L2TP VPN.
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC1 -p tcp --dport 88 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC1 -p tcp --sport 88 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC1 -p tcp --dport 135 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC1 -p tcp --sport 135 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC1 -p tcp --dport 139 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC1 -p tcp --sport 139 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC1 -p tcp --dport 389 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC1 -p tcp --sport 389 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC1 -p tcp --dport 445 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC1 -p tcp --sport 445 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC2 -p tcp --dport 88 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC2 -p tcp --sport 88 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC2 -p tcp --dport 135 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC2 -p tcp --sport 135 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC2 -p tcp --dport 139 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC2 -p tcp --sport 139 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC2 -p tcp --dport 389 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC2 -p tcp --sport 389 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC2 -p tcp --dport 445 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC2 -p tcp --sport 445 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC3 -p tcp --dport 88 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC3 -p tcp --sport 88 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC3 -p tcp --dport 135 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC3 -p tcp --sport 135 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC3 -p tcp --dport 139 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC3 -p tcp --sport 139 \
-m state --state ESTABLISHED,RELATED -j ACCEPT

```



```

$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC3 -p tcp --dport 389 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC3 -p tcp --sport 389 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC3 -p tcp --dport 445 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC3 -p tcp --sport 445 \
-m state --state ESTABLISHED,RELATED -j ACCEPT

$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC1 -p tcp \
--dport 5555:5655 -m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC1 -p tcp \
--sport 5555:5655 -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC2 -p tcp \
--dport 5555:5655 -m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC2 -p tcp \
--sport 5555:5655 -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC3 -p tcp \
--dport 5555:5655 -m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC3 -p tcp \
--sport 5555:5655 -m state --state ESTABLISHED,RELATED -j ACCEPT

# Enable the L2TP to update its AntiVirus software. (We won't use SUS for this
# server, would rather do it manually to ensure nothing breaks)
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_NAV -p tcp --dport 21 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_NAV -p tcp --sport 21 \
-m state --state ESTABLISHED,RELATED -j ACCEPT

# L2TP VPN Traffic with the Exchange server. Again, this is explained in the
# VPN text below; we'll log most of this:
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_SMTP -p tcp --dport 4444 \
-m state --state NEW -j LOG --log-level debug --log-prefix "EXCH SA: "
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_SMTP -p tcp --dport 4444 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_SMTP -p tcp --dport 4445 \
-m state --state NEW -j LOG --log-level debug --log-prefix "EXCH NSPI: "
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_SMTP -p tcp --dport 4445 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_SMTP -p tcp --dport 4446 \
-m state --state NEW -j LOG --log-level debug --log-prefix "EXCH IS: "
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_SMTP -p tcp --dport 4446 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_SMTP -p tcp \
--dport 5555:5655 -m state --state NEW -j ACCEPT

# Replies from Exchange to L2TP:
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_SMTP -p tcp --sport 4444 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_SMTP -p tcp --sport 4445 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_SMTP -p tcp --sport 4446 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_SMTP -p tcp \
--sport 5555:5655 -m state --state ESTABLISHED,RELATED -j ACCEPT

# Other traffic to Exchange

```

```

$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC3 -p tcp --dport 88 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC3 -p tcp --sport 88 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC3 -p tcp --dport 135 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC3 -p tcp --sport 135 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A TCP -i $VPN_INT_IF -o $INT_IF -d $INT_DC3 -p tcp --dport 445 \
-m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $VPN_INT_IF -s $INT_DC3 -p tcp --sport 445 \
-m state --state ESTABLISHED,RELATED -j ACCEPT

# Next, we'll enable outbound passive FTP. This is similar to the Web
# traffic.
# Note that we're also using the Squid proxy's firewall to restrict which
# internal machines have access to FTP.
# We'll log this. Note that when we review our logs, there should be
# corresponding entries for both trips through the firewall, in both
# directions; if not, something odd's going on!

# Traffic from inside to the DMZ proxy:
$IPTABLES -A TCP -i $INT_IF -o $DMZ_IF -s $INT_IT_NET -d $DMZ_SQUID \
-p tcp --dport 21 -m state --state NEW -j LOG --log-level debug \
--log-prefix "FTP : "
$IPTABLES -A TCP -i $INT_IF -o $DMZ_IF -s $INT_IT_NET -d $DMZ_SQUID \
-p tcp --dport 21 -m state --state NEW -j ACCEPT

# FTP traffic from the proxy to the Internet:
$IPTABLES -A TCP -i $DMZ_IF -o $EXT_IF -s $DMZ_SQUID -p tcp --dport 21 \
-m state --state NEW -j LOG --log-level debug --log-prefix "FTP : "
$IPTABLES -A TCP -i $DMZ_IF -o $EXT_IF -s $DMZ_SQUID -p tcp --dport 21 \
-m state --state NEW -j ACCEPT

# FTP traffic from the Internet back to the proxy (valid replies only!)
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -d $DMZ_SQUID -p tcp --sport 21 \
-m state --state ESTABLISHED,RELATED -j LOG --log-level debug \
--log-prefix "FTP : "
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -d $DMZ_SQUID -p tcp --sport 21 \
-m state --state ESTABLISHED,RELATED -j ACCEPT

# FTP traffic from the proxy back to the inside (valid replies only!)
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_SQUID -d $INT_IT_NET \
-p tcp --sport 21 -m state --state ESTABLISHED,RELATED -j LOG \
--log-level debug --log-prefix "FTP : "
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_SQUID -d $INT_IT_NET \
-p tcp --sport 21 -m state --state ESTABLISHED,RELATED -j ACCEPT

# Last FTP we need is our DMZ NAV server to the outside for updates
$IPTABLES -A TCP -i $DMZ_IF -o $EXT_IF -s $DMZ_NAV -p tcp --dport 21 -m state \
--state NEW -j ACCEPT
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -d $DMZ_NAV -p tcp --sport 21 -m state \
--state ESTABLISHED,RELATED -j ACCEPT

#
# Our next step is to enable SSH. This, too, must go through a proxy in the
# DMZ. Unlike our FTP, though, this is enabled in both directions. We do want

```

```

# to log all of this.

# First, we set up the outbound traffic.

# SSH traffic from inside to the DMZ proxy; this is only allowed from the IT
# staff's computers:
$IPTABLES -A TCP -i $INT_IF -o $DMZ_IF -s $INT_IT_NET -d $DMZ_SSH \
  -p tcp --dport 22 -m state --state NEW -j LOG --log-level debug \
  --log-prefix "SSH OUT: "
$IPTABLES -A TCP -i $INT_IF -o $DMZ_IF -s $INT_IT_NET -d $DMZ_SSH \
  -p tcp --dport 22 -m state --state NEW -j ACCEPT

# SSH traffic from the proxy to the Internet:
$IPTABLES -A TCP -i $DMZ_IF -o $EXT_IF -s $DMZ_SSH -p tcp --dport 22 \
  -m state --state NEW -j LOG --log-level debug --log-prefix "SSH OUT: "
$IPTABLES -A TCP -i $DMZ_IF -o $EXT_IF -s $DMZ_SSH -p tcp --dport 22 \
  -m state --state NEW -j ACCEPT

# SSH traffic from the Internet back to the proxy (valid replies only!)
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -d $DMZ_SSH -p tcp --sport 22 \
  -m state --state ESTABLISHED,RELATED -j LOG --log-level debug \
  --log-prefix "SSH OUT-REPLY: "
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -d $DMZ_SSH -p tcp --sport 22 \
  -m state --state ESTABLISHED,RELATED -j ACCEPT

# SSH traffic from the proxy back to the inside (valid replies only!)
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_SSH -d $INT_IT_NET \
  -p tcp --sport 22 -m state --state ESTABLISHED,RELATED -j LOG \
  --log-level debug --log-prefix "SSH OUT-REPLY: "
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_SSH -d $INT_IT_NET \
  -p tcp --sport 22 -m state --state ESTABLISHED,RELATED -j ACCEPT

# Next, we set up the ability for IT staff to SSH in from remote locations; we
# don't expect this to happen very often, though the likelihood of needing SSH
# always seems higher when IT staff are out of the office...

# SSH traffic from outside to the DMZ
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -d $DMZ_SSH -p tcp --dport 22 \
  -m state --state NEW -j LOG --log-level debug --log-prefix \
  "SSH IN: "
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -d $DMZ_SSH -p tcp --dport 22 \
  -m state --state NEW -j ACCEPT

# SSH traffic from the DMZ to the internal network; since our staffs' Windows
# clients don't have SSH installed, this should only be going to servers:
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_SSH -d $INT_VIP -p tcp \
  --dport 22 -m state --state NEW -j LOG --log-level warn --log-prefix \
  "SSH IN: "
$IPTABLES -A TCP -i $DMZ_IF -o $INT_IF -s $DMZ_SSH -d $INT_VIP -p tcp \
  --dport 22 -m state --state NEW -j ACCEPT

# SSH traffic from the inside back to the DMZ (valid replies only!) This
# should only be coming from servers...
$IPTABLES -A TCP -i $INT_IF -o $DMZ_IF -s $INT_VIP -d $DMZ_SSH -p tcp \
  --sport 22 -m state --state ESTABLISHED,RELATED -j LOG --log-level \
  debug --log-prefix "SSH IN-REPLY: "
$IPTABLES -A TCP -i $INT_IF -o $DMZ_IF -s $INT_VIP -d $DMZ_SSH -p tcp \
  --sport 22 -m state --state ESTABLISHED,RELATED -j ACCEPT

```

```

# SSH traffic from the DMZ back to the Internet (valid replies only!)
$IPTABLES -A TCP -i $DMZ_IF -o $EXT_IF -s $DMZ_SSH -p tcp --sport 22 \
    -m state --state ESTABLISHED,RELATED -j LOG --log-level debug \
    --log-prefix "SSH IN-REPLY: "
$IPTABLES -A TCP -i $DMZ_IF -o $EXT_IF -s $DMZ_SSH -p tcp --sport 22 \
    -m state --state ESTABLISHED,RELATED -j ACCEPT

# We also want to allow SSH to the Firewall itself
$IPTABLES -A TCP -i $DMZ_IF -s $DMZ_SSH -d $DMZ_FW -p tcp --dport 22 \
    -m state --state NEW -j LOG --log-level warn --log-prefix "SSH TO FW: "
$IPTABLES -A TCP -i $DMZ_IF -s $DMZ_SSH -d $DMZ_FW -p tcp --dport 22 \
    -m state --state NEW -j ACCEPT
$IPTABLES -A TCP -o $DMZ_IF -s $DMZ_FW -d $DMZ_SSH -p tcp --sport 22 \
    -m state --state ESTABLISHED,RELATED -j LOG --log-level debug \
    --log-prefix "SSH TO FW-REPLY: "
$IPTABLES -A TCP -o $DMZ_IF -s $DMZ_FW -d $DMZ_SSH -p tcp --sport 22 \
    -m state --state ESTABLISHED,RELATED -j ACCEPT

# We also need to enable our PPTP sessions to the Netopia border router. Note
# that while the telnet we use to talk with the Netopia R5300 will be
# tunneled inside the PPTP session so we don't need to tell the firewall
# about it. Lastly, note that in accordance with our policy of not letting
# traffic directly between the internal network and the Internet, this will be
# done by first using SSH to the DMZ server (see above) and then establishing
# the PPTP session to the border router from there.

$IPTABLES -A TCP -i $DMZ_IF -o $EXT_IF -s $DMZ_SSH -d $EXT_ROUTER -p tcp \
    --dport 1723 -m state --state NEW -j LOG --log-level debug --log-prefix \
    "PPTP TO ROUTER: "
$IPTABLES -A TCP -i $DMZ_IF -o $EXT_IF -s $DMZ_SSH -d $EXT_ROUTER -p tcp \
    --dport 1723 -m state --state NEW -j ACCEPT
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -s $EXT_ROUTER -d $DMZ_SSH -p tcp \
    --sport 1723 -m state --state ESTABLISHED,RELATED -j LOG \
    --log-level debug --log-prefix "PPTP FROM ROUTER: "
$IPTABLES -A TCP -i $EXT_IF -o $DMZ_IF -s $EXT_ROUTER -d $DMZ_SSH -p tcp \
    --sport 1723 -m state --state ESTABLISHED,RELATED -j ACCEPT

# Lastly, we send RST's to Ident traffic as a nice neighbor; no sense making
# them wait.
$IPTABLES -A TCP -i $EXT_IF -p tcp --dport 113 -j REJECT --reject-with \
    tcp-reset

# If we get here, we want to know what's going on; it's not something we want,
# so log it & then drop it.
$IPTABLES -A TCP -j LOG --log-level debug --log-prefix "DISALLOWED TCP: "
$IPTABLES -A TCP -j DROP

```

```

####
#
# UDP chain - filters all udp traffic (IP protocol 17)
#
# Obviously, we allow only the UDP we want, according to our security policy
# -- which is outbound DNS & NTP and their replies; Syslog; and L2TP & IKE
# NOTE: though it sounds strange, Netfilter actually lets us track "state" of
# UDP to some extent

```

```

# First, allow DNS queries. This goes from our internal network, to the
# caching DNS server in the DMZ, and from there to the Internet -- and then
# back again.
# There's too much traffic to log all of this. (Note: we're only allowing
# udp, since we don't want zone transfers. This isn't compliant with the DNS
# specifications; large replies will try to use TCP and fail. That shouldn't
# be a problem for us, though; if anything, blocking such large DNS replies
# may help block would-be cache poisoning. See RFC 1035, which describes
# DNS, at http://www.ietf.org/rfc/rfc1035.txt
#
# Too much traffic here to log...

# DNS queries out to the Internet via the DMZ:
$IPTABLES -A UDP -i $INT_IF -o $DMZ_IF -s $INT_DNS -d $DMZ_DNS -p udp \
    --dport 53 -m state --state NEW -j ACCEPT
$IPTABLES -A UDP -i $DMZ_IF -o $EXT_IF -s $DMZ_DNS -p udp --dport 53 \
    -m state --state NEW -j ACCEPT

# DNS replies back from the Internet
$IPTABLES -A UDP -i $EXT_IF -o $DMZ_IF -d $DMZ_DNS -p udp --sport 53 \
    -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A UDP -i $DMZ_IF -o $INT_IF -s $DMZ_DNS -d $INT_DNS -p udp \
    --sport 53 -m state --state ESTABLISHED,RELATED -j ACCEPT

# DNS queries by our L2TP server & clients:
$IPTABLES -A UDP -i $VPN_INT_IF -o $INT_IF -d $INT_DNS -p udp --dport 53 \
    -m state --state NEW -j ACCEPT
$IPTABLES -A UDP -i $INT_IF -o $VPN_INT_IF -s $INT_DNS -p udp --sport 53 \
    -m state --state ESTABLISHED,RELATED -j ACCEPT

#
# Let the firewall send syslog to DMZ_SYSLOG server
# Note: This could also go in the OUTPUT chain

$IPTABLES -A UDP -o $DMZ_IF -s $DMZ_FW -d $DMZ_SYSLOG -p udp --dport 514 \
    -m state --state NEW -j ACCEPT

# Let the Border Router send syslog info to the DMZ syslog server
$IPTABLES -A UDP -i $EXT_IF -o $DMZ_IF -s $EXT_ROUTER -d $DMZ_SYSLOG -p udp \
    --dport 514 -m state --state NEW -j ACCEPT

# Let the L2TP VPN server's NTSyslog send info to the DMZ syslog server
$IPTABLES -A UDP -i $VPN_INT_IF -o $DMZ_IF -s $DMZ_L2TP -d $DMZ_SYSLOG \
    -p udp --dport 514 -m state --state NEW -j ACCEPT

# Let the DMZ's syslog server send info to the Internal syslog server
$IPTABLES -A UDP -i $DMZ_IF -o $INT_IF -s $DMZ_SYSLOG -d $INT_SYSLOG -p udp \
    --dport 514 -m state --state NEW -j ACCEPT

# NTP, so we can keep things synchronized
#
# First, let the internal time server get to the DMZ's time server
$IPTABLES -A UDP -i $INT_IF -o $DMZ_IF -s $INT_NTP -d $DMZ_NTP -p udp \
    --dport 123 -m state --state NEW -j ACCEPT

# Next, let the DMZ time server get to the server on the Internet

```

```

$IPTABLES -A UDP -i $DMZ_IF -o $EXT_IF -s $DMZ_NTP -d $PUB_NTP1 -p udp \
--dport 123 -m state --state NEW -j ACCEPT
$IPTABLES -A UDP -i $DMZ_IF -o $EXT_IF -s $DMZ_NTP -d $PUB_NTP2 -p udp \
--dport 123 -m state --state NEW -j ACCEPT

# Also let our L2TP server synchronize
$IPTABLES -A UDP -i $VPN_INT_IF -o $DMZ_IF -s $DMZ_L2TP -d $DMZ_NTP -p udp \
--dport 123 -m state --state NEW -j ACCEPT

# Now we need to let the NTP answers back in
# From the Internet to the DMZ
$IPTABLES -A UDP -i $EXT_IF -o $DMZ_IF -s $PUB_NTP1 -d $DMZ_NTP -p udp \
--sport 123 -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A UDP -i $EXT_IF -o $DMZ_IF -s $PUB_NTP2 -d $DMZ_NTP -p udp \
--sport 123 -m state --state ESTABLISHED,RELATED -j ACCEPT

# From the DMZ to the inside
$IPTABLES -A UDP -i $DMZ_IF -o $INT_IF -s $DMZ_NTP -d $INT_NTP -p udp \
--sport 123 -m state --state ESTABLISHED,RELATED -j ACCEPT

# From the DMZ back to the L2TP server:
$IPTABLES -A UDP -i $DMZ_IF -o $VPN_INT_IF -s $DMZ_NTP -d $DMZ_L2TP -p udp \
--sport 123 -m state --state ESTABLISHED,RELATED -j ACCEPT

# Now, we need to add in our L2TP VPN's UDP requirements

# Kerberos
$IPTABLES -A UDP -i $VPN_INT_IF -o $INT_IF -d $INT_DC1 -p udp --dport 88 \
-m state --state NEW -j ACCEPT
$IPTABLES -A UDP -i $VPN_INT_IF -o $INT_IF -d $INT_DC2 -p udp --dport 88 \
-m state --state NEW -j ACCEPT
$IPTABLES -A UDP -i $VPN_INT_IF -o $INT_IF -d $INT_DC3 -p udp --dport 88 \
-m state --state NEW -j ACCEPT
$IPTABLES -A UDP -i $VPN_INT_IF -o $INT_IF -d $INT_SMTP -p udp --dport 88 \
-m state --state NEW -j ACCEPT
$IPTABLES -A UDP -i $INT_IF -o $VPN_INT_IF -s $INT_DC1 -p udp --sport 88 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A UDP -i $INT_IF -o $VPN_INT_IF -s $INT_DC2 -p udp --sport 88 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A UDP -i $INT_IF -o $VPN_INT_IF -s $INT_DC3 -p udp --sport 88 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A UDP -i $INT_IF -o $VPN_INT_IF -s $INT_SMTP -p udp --sport 88 \
-m state --state ESTABLISHED,RELATED -j ACCEPT

# LDAP
$IPTABLES -A UDP -i $VPN_INT_IF -o $INT_IF -d $INT_DC1 -p udp --dport 389 \
-m state --state NEW -j ACCEPT
$IPTABLES -A UDP -i $INT_IF -o $VPN_INT_IF -s $INT_DC1 -p udp --sport 389 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A UDP -i $VPN_INT_IF -o $INT_IF -d $INT_DC2 -p udp --dport 389 \
-m state --state NEW -j ACCEPT
$IPTABLES -A UDP -i $INT_IF -o $VPN_INT_IF -s $INT_DC2 -p udp --sport 389 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A UDP -i $VPN_INT_IF -o $INT_IF -d $INT_DC3 -p udp --dport 389 \
-m state --state NEW -j ACCEPT
$IPTABLES -A UDP -i $INT_IF -o $VPN_INT_IF -s $INT_DC3 -p udp --sport 389 \

```

```

-m state --state ESTABLISHED,RELATED -j ACCEPT

# IKE, needed for IPSec
$IPTABLES -A UDP -i $EXT_IF -o $VPN_EXT_IF -d $PUB_L2TP -p udp --dport 500 \
-m state --state NEW -j LOG --log-level debug --log-prefix "IKE: "
$IPTABLES -A UDP -i $EXT_IF -o $VPN_EXT_IF -d $PUB_L2TP -p udp --dport 500 \
-m state --state NEW -j ACCEPT
$IPTABLES -A UDP -i $VPN_EXT_IF -o $EXT_IF -s $PUB_L2TP -p udp --sport 500 \
-m state --state ESTABLISHED,RELATED -j ACCEPT

# L2TP itself
$IPTABLES -A UDP -i $EXT_IF -o $VPN_EXT_IF -d $PUB_L2TP -p udp --dport 1701 \
-m state --state NEW -j LOG --log-level debug --log-prefix "L2TP: "
$IPTABLES -A UDP -i $EXT_IF -o $VPN_EXT_IF -d $PUB_L2TP -p udp --dport 1701 \
-m state --state NEW -j ACCEPT
$IPTABLES -A UDP -i $VPN_EXT_IF -o $EXT_IF -s $PUB_L2TP -p udp --sport 1701 \
-m state --state ESTABLISHED,RELATED -j ACCEPT

# NetBIOS & Login -- in theory we think Win2k should work without this, but not
# in practice

$IPTABLES -A UDP -i $EXT_IF -o $VPN_EXT_IF -d $PUB_L2TP -p udp --dport 137 \
-m state --state NEW -j LOG --log-level debug --log-prefix "137: "
$IPTABLES -A UDP -i $EXT_IF -o $VPN_EXT_IF -d $PUB_L2TP -p udp --dport 137 \
-m state --state NEW -j ACCEPT
$IPTABLES -A UDP -i $VPN_EXT_IF -o $EXT_IF -s $PUB_L2TP -p udp --sport 137 \
-m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A UDP -i $EXT_IF -o $VPN_EXT_IF -d $PUB_L2TP -p udp --dport 138 \
-m state --state NEW -j LOG --log-level debug --log-prefix "138: "
$IPTABLES -A UDP -i $EXT_IF -o $VPN_EXT_IF -d $PUB_L2TP -p udp --dport 138 \
-m state --state NEW -j ACCEPT
$IPTABLES -A UDP -i $VPN_EXT_IF -o $EXT_IF -s $PUB_L2TP -p udp --sport 138 \
-m state --state ESTABLISHED,RELATED -j ACCEPT

# We also want IKE for IPSec with the DMZ - these communications must come
# from the inside, not the DMZ.
$IPTABLES -A UDP -i $INT_IF -o $DMZ_IF -p udp --dport 500 \
-m state --state NEW -j ACCEPT
$IPTABLES -A UDP -i $DMZ_IF -o $INT_IF -p udp --sport 500 \
-m state --state ESTABLISHED,RELATED -j ACCEPT

# Drop all other UDP traffic; because Windows machines are so chatty (e.g. UDP
# 137) we'll only log stuff coming into the External and DMZ interfaces.
#

$IPTABLES -A UDP -i $EXT_IF -p udp -j LOG --log-level warn \
--log-prefix "DISALLOWED UDP: "
$IPTABLES -A UDP -i $DMZ_IF -p udp -j LOG --log-level warn \
--log-prefix "DISALLOWED UDP: "
$IPTABLES -A UDP -p udp -j DROP

```

```

####
#
# ICMP chain - filters all icmp traffic (IP protocol 1)

```

```
#
# We only allow the stuff we really need

# We want to be able to PING
$IPTABLES -A ICMP -o $EXT_IF -s $INT_NET -p icmp --icmp-type 8 -m state \
--state NEW -j ACCEPT
$IPTABLES -A ICMP -i $EXT_IF -d $INT_NET -p icmp --icmp-type 0 -m state \
--state ESTABLISHED,RELATED -j ACCEPT

#
# Destination Unreachables, so we don't have to wait to time out
#
$IPTABLES -A ICMP -i $EXT_IF -d $INT_NET -p icmp --icmp-type 3 -m state \
--state ESTABLISHED,RELATED -j ACCEPT
#
# Let the outside world think they're able to ping our public addresses
#
$IPTABLES -A ICMP -i $EXT_IF -p icmp --icmp-type 8 -d $PUB_NET -m state \
--state NEW -j REJECT --reject-with icmp-host-prohibited

# Log & Drop all the rest
$IPTABLES -A ICMP -p icmp -j LOG --log-level warn --log-prefix "DENIED ICMP: "
$IPTABLES -A ICMP -p icmp -j DROP
```

```
#####
#
# OTHER chain - this filters traffic of any other IP protocol
#
# We need ESP (type 50) for our L2TP/IPSec VPN, and for some of our traffic
# between the DMZ and the inside
#
# Note that there's no point enabling it from the DMZ to the Internet, since
# our NAT will break IPSec anyway!

# First, allow establishment between outside and L2TP VPN
$IPTABLES -A OTHER -i $EXT_IF -o $VPN_EXT_IF -p 50 -j LOG --log-level debug \
--log-prefix "ESP: "
$IPTABLES -A OTHER -i $EXT_IF -o $VPN_EXT_IF -p 50 -j ACCEPT
$IPTABLES -A OTHER -i $VPN_EXT_IF -o $EXT_IF -p 50 -j LOG --log-level debug \
--log-prefix "ESP: "
$IPTABLES -A OTHER -i $VPN_EXT_IF -o $EXT_IF -p 50 -j ACCEPT

# Next, enable it between internal network and DMZ; since this comprises a
# huge amount of traffic, we won't bother to log it
$IPTABLES -A OTHER -i $DMZ_IF -o $INT_IF -p 50 -j ACCEPT
$IPTABLES -A OTHER -i $INT_IF -o $DMZ_IF -p 50 -j ACCEPT

# We also want to use AH for communication with the DMZ servers
$IPTABLES -A OTHER -i $DMZ_IF -o $INT_IF -p 51 -j ACCEPT
$IPTABLES -A OTHER -i $INT_IF -o $DMZ_IF -p 51 -j ACCEPT

# We also need PPTP, but only from DMZ to the border router & back
$IPTABLES -A OTHER -p 47 -i $DMZ_IF -o $EXT_IF -s $DMZ_SSH -d $EXT_ROUTER \
-j LOG --log-level debug --log-prefix "PPTP: "
$IPTABLES -A OTHER -p 47 -i $DMZ_IF -o $EXT_IF -s $DMZ_SSH -d $EXT_ROUTER \
-j ACCEPT
$IPTABLES -A OTHER -p 47 -i $EXT_IF -o $DMZ_IF -s $EXT_ROUTER -d $DMZ_SSH \
```



```

-j LOG --log-level debug --log-prefix "PPTP: "
$IPTABLES -A OTHER -p 47 -i $EXT_IF -o DMZ_IF -s $EXT_ROUTER -d $DMZ_SSH \
-j ACCEPT

# Drop any other type of traffic
#
$IPTABLES -A OTHER -j LOG --log-level warn --log-prefix "ODD PROTOCOL: "
$IPTABLES -A OTHER -j DROP

```

```

#####
#
# OUTPUT chain - this filters traffic generated by the firewall itself
#
# As with the INPUT chain, we could filter things here -- the only thing that
# should be coming out is syslog, NTP, & SSH replies -- but we put it in the
# TCP & UDP chains for cross-checking purposes
#
# Send everything to the appropriate chain
$IPTABLES -A OUTPUT -p tcp -j TCP
$IPTABLES -A OUTPUT -p udp -j UDP
$IPTABLES -A OUTPUT -p icmp -j ICMP
$IPTABLES -A OUTPUT -j OTHER

# Log & drop everything that gets returned to here...

$IPTABLES -A OUTPUT -j LOG --log-level warn --log-prefix "FW COMPROMISE?: "
$IPTABLES -A OUTPUT -j DROP

```

```

#####
#
# POSTROUTING chain -- this just Source NAT's our outbound traffic, since our
# private addresses aren't routable and we need a valid public address to
# communicate with the outside world. We do no filtering here, that was done
# in the "filter" table already; now we're in the "nat" table, "it should
# only be used to translate the packet's source field or destination field"1
# We SNAT everything headed to the Internet except our L2TP traffic
$IPTABLES -t nat -A POSTROUTING -s $INT_NET -j SNAT \
--to-source $SNAT_IP

```

# This concludes our ruleset creation! That wasn't so bad, now, was it? ;^)

```

####
#
# Kernel preparations, now that Netfilter is up & running.
#
# Note that these could also be put into a different startup script entirely
#

```

```

touch /var/lock/subsys/local
# Make sure the firewall responds to NAT replies
/sbin/ifconfig eth0:0 5.6.7.7 netmask 255.255.255.240

# Drop ICMP broadcasts in kernel
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

```

```

# Drop source-routed packets in kernel
for f in /proc/sys/net/ipv4/conf/*/accept_source_route; do
    echo 0 > $f
done

# Drop ICMP redirect packets in kernel
for f in /proc/sys/net/ipv4/conf/*/accept_redirects; do
    echo 0 > $f
done

# Do not create ICMP redirect packets in kernel
for f in /proc/sys/net/ipv4/conf/*/send_redirects; do
    echo 0 > $f
done

# Just in case this didn't happen elsewhere already,
# enable IP forwarding so we can route
echo 1 > /proc/sys/net/ipv4/ip_forward

# Set up our routing table
/sbin/route add -net 5.6.7.0 netmask 255.255.255.240 gw 5.6.7.9
/sbin/route add 5.6.7.8 gw 5.6.7.2

```

---

The final preparatory step is to burn our pre-configured, pre-hardened operating system onto a bootable CD. This is so that it is not possible for anyone without physical access to alter the startup routine in any way. One of the first actions taken in the event of a successful compromise is often to force the machine to reboot in order to alter the programs launched or installed during bootstrapping. Because the attacker will have no means to alter the files on the CD, a reboot will simply return the firewall to its desired state. (Note that the CD-ROM drive used for booting the firewall lacks the capability to write to CD.)

An explanation of how to make a bootable Linux firewall CD is beyond the scope of this document. See <http://rr.sans.org/linux/cdrom.php> for a nice summary (note: this site requires free registration.) There is also an intriguing project at <http://cd-linux.org/build-gs.htm> with a goal of helping people make CD-based Linux systems. And, of course, there's also the ubiquitous Linux HOWTO's; see the Boot Disk howto, which contains a section dealing specifically with CD's at: <http://www.linux.org/docs/ldp/howto/Bootdisk-HOWTO/cd-roms.html>.

## B. Firewall B:

We also have a separate firewall for guest Internet access in the conference room. This is a much simpler configuration. Note that the basics of preparing and hardening the machine are the same for this machine as for the primary firewall. Rather than repeat all of that, we will simply list the different firewall configuration script here:

```
#####
```

```

#
# GIAC Enterprises v.1.7 Netfilter Conference Room Firewall script
#
# Written by Vincent R. Streiff, Fall 2002
#
# This could not have been possible without guidance from the iptables
# tutorial written by Oskar Andreason, located at:
# http://www.netfilter.org/documentation/tutorials/blueflux/
#
# A plethora of other tutorials which may help can be found at:
# http://www.netfilter.org/documentation/index.html#HOWTO
#
#
# Our general guidelines:
# Maximize performance, but not at the expense of security
# Since complexity is bad, there may be instances where script
# legibility takes precedence over streamlining for performance
#
#####

#####
#
# For starters, to make all this legible and editable, we set our
# variables used in the script
#

IPTABLES="/user/sbin/iptables # path to the iptables command; your mileage may
                                # vary depending on where you put it.
EXT_IF="eth0" # This is the interface facing the Internet,
              # connected to our router
INT_IF="eth1" # This is the interface facing the internal network
EXT_ROUTER="5.6.7.1" # The address of our Netopia R5300 router
PUB_FW="5.6.7.13" # Public address of our firewall (eth0)
INT_FW="10.99.99.1" # IP address of firewall's internal NIC (eth1)
PUB_NTP1="5.127.125.23" # Not really of course; you can often use ISP's routers.
PUB_NTP2="5.118.34.135" # We're using "stratum 2" servers -- after notifying
                        # the admins of those servers, of course -- to keep
                        # ourselves as accurate as possible. We use 2 NTP
                        # servers for redundancy; note that coordination with
                        # the "official" time isn't as critical as keeping
                        # all of our servers synchronized with each other.
                        # See www.ntp.org for more on Network Time Protocol
GIAC_SNAT_IP="5.6.7.7" # IP address used by the GIAC offices for NAT. We
                       # only allow SSH to this firewall from that address
SNAT_IP="5.6.7.14" # IP address used for NAT'ing our public traffic

###
#
# Make sure Netfilter is pristine before we start
#

## First, we flush things out, just to be safe
#
$IPTABLES -F
$IPTABLES -X # Delete any existing custom chains before we start
$IPTABLES -t nat -F
$IPTABLES -t mangle -F # We aren't using this table, but still want it clean

```

```
##
# Next, we set the default policies for our tables; we drop by default
# according to our policy of least required privilege; we only want traffic
# we specifically allow to get through.
$IPTABLES -P INPUT DROP      # Default policy of Drop
$IPTABLES -P FORWARD DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -t nat -P PREROUTING ACCEPT # We want to avoid filtering here17
$IPTABLES -t nat -P POSTROUTING ACCEPT # No filtering here, that's been done
$IPTABLES -t nat -P OUTPUT ACCEPT
$IPTABLES -t mangle -P PREROUTING ACCEPT # We avoid filtering here
$IPTABLES -t mangle -P INPUT ACCEPT
$IPTABLES -t mangle -P FORWARD ACCEPT
$IPTABLES -t mangle -P OUTPUT ACCEPT
$IPTABLES -t mangle -P POSTROUTING ACCEPT
```

```
###
```

```
#
# Kernel preparations:
# Most scripts have them at the beginning, we're putting them at the
# end to avoid the momentary exposure -- Netfilter lets us set rules
# before interfaces even exist, so we can wait until it's ready before
# we turn on any routing
#
```

```
# Now we add our custom chains.
```

```
# For the filter table -- i.e. INPUT, FORWARD, & OUTPUT
$IPTABLES -t filter -N TCP_FLAG
$IPTABLES -t filter -N SOURCE_CHECK
$IPTABLES -t filter -N EX_SOURCE_CHECK
$IPTABLES -t filter -N ICMP
$IPTABLES -t filter -N TCP
$IPTABLES -t filter -N UDP
$IPTABLES -t filter -N OTHER
```

```
# PREROUTING chain isn't used, since we aren't providing any services
```

```
####
```

```
#
# INPUT chain -- any traffic going TO the firewall itself
#
# First, check for invalid flags & invalid sources
# We check for invalid flags first for logging purposes -- assume
# most would also be caught with source_check, but
# want to be able to find them in the logs!
#
$IPTABLES -A INPUT -p tcp -j TCP_FLAG
$IPTABLES -A INPUT -j SOURCE_CHECK
$IPTABLES -A INPUT -i $EXT_IF -j EX_SOURCE_CHECK
# Note on EX_SOURCE_CHECK: all traffic coming into the EXT_IF interface will get
# dropped anyway, but we still want to check it for logging purposes. Note also
# that if we're seeing invalid sources here, it means our border router's
```

<sup>17</sup> See <http://www.netfilter.org/documentation/tutorials/blueflux/iptables-tutorial.html#TABLES> which states in part, "The rest of the packets in the stream will... not go through this table again, but instead they will automatically have the same actions taken to them as the first packet in the stream. This is one reason why you should not do any filtering in this table..."

```
# filtering has failed!!

# We don't actually allow ANY direct access to the firewall from the Internet or
# the internal network, so if it hasn't been dropped & logged already,
# do so now
#
$IPTABLES -A INPUT -i $EXT_IF -j LOG --log-level warn -j log-prefix \
    "FW ACCESS ATTEMPT: "
$IPTABLES -A INPUT -i $EXT_IF -j DROP
$IPTABLES -A INPUT -i $INT_IF -j LOG --log-level warn -j log-prefix \
    "FW ACCESS ATTEMPT: "
$IPTABLES -A INPUT -i $INT_IF -j DROP

# Now we analyze -- it would be easy to just filter everything we want
# right here, since the only traffic we allow to the firewall itself is SSH
# from DMZ & NTP replies. It'll be easier to cross-check open ports later,
# though, if we check it in the TCP & UDP chains. Note also that doing it here
# instead would be more efficient; we don't expect a performance problem, though,
# so this shouldn't be an issue.

$IPTABLES -A INPUT -p tcp -j TCP
$IPTABLES -A INPUT -p udp -j UDP

# Log & drop anything else that made it this far through the INPUT chain
#
$IPTABLES -A INPUT -j LOG --log-level warn --log-prefix "FW ACCESS ATTEMPT: "
$IPTABLES -A INPUT -j DROP
```

```
#####
#
# FORWARD chain -- filters anything going THROUGH the firewall
#
# First, check for invalid flags & invalid sources.
# We check for invalid flags first for logging purposes -- assume
# most would probably be caught with source_check, but
# want to be able to find them in the logs!
#
$IPTABLES -A FORWARD -p tcp -j TCP_FLAG
$IPTABLES -A FORWARD -j SOURCE_CHECK
$IPTABLES -A FORWARD -i $EXT_IF -j EX_SOURCE_CHECK
# Note that if we're seeing invalid sources here, it means our border router's
# filtering has failed!!

#
# If the traffic made it through, we check to see if we want it. We break this
# down by protocol so it's easier to read; plus, we can list rules by protocol
# to ease cross-checking later, after we're up & running
#
# Order is important here. We put the jump to TCP first, since that's the bulk
# of our traffic; the jump to the OTHER chain goes at the end because traffic
# for tcp, udp, and icmp will already be filtered out, leaving a couple fewer
# protocols. If, for some reason, traffic comes back here after jumping to one
# of the first three chains, it will go through the OTHER chain, but that really
# doesn't hurt anything; it's just one more place to drop
```

```
# it...

$IPTABLES -A FORWARD -p tcp -j TCP
$IPTABLES -A FORWARD -p udp -j UDP
$IPTABLES -A FORWARD -p icmp -j ICMP
$IPTABLES -A FORWARD -j OTHER

# If it made it back here somehow, log for analysis & drop
#
$IPTABLES -A FORWARD -j LOG --log-level warn --log-prefix "FUNKY TRAFFIC: "
$IPTABLES -A FORWARD -j DROP
```

```
####
# TCP_FLAG Chain
#
# Checks for invalid TCP Flags
#
#

$IPTABLES -A TCP_FLAG -p tcp ! --syn -m state --state NEW -j LOG \
    --log-level warn --log-prefix "NEW not SYN: "
$IPTABLES -A TCP_FLAG -p tcp ! --syn -m state --state NEW -j DROP
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags ALL NONE -j LOG --log-level warn \
    --log-prefix "TCP FLAGS = NONE: "
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags ALL NONE -j DROP

#
# We check for all-flag scans before the more specific ones, otherwise log
# prefixes could be misleading. This is because we tell iptables which flags to
# examine, followed by which flags to match. A packet with all flags on
# would give us a false positive hit on all of the following rules...
#
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags ALL ALL -j LOG --log-level warn \
    --log-prefix "TCP FLAGS = ALL: "
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags ALL ALL -j DROP
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags FIN,PSH,URG FIN,PSH,URG -j LOG \
    --log-level warn --log-prefix "TCP FLAGS = CHRISTMAS: "
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags FIN,PSH,URG FIN,PSH,URG -j DROP
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags FIN,SYN FIN,SYN -j LOG \
    --log-level warn --log-prefix "TCP FLAGS = SYN-FIN: "
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags FIN,SYN FIN,SYN -j DROP
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags SYN,RST SYN,RST -j LOG \
    --log-level warn --log-prefix "TCP FLAGS = SYN-RST: "
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags SYN,RST SYN,RST -j DROP
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags FIN,RST FIN,RST -j LOG \
    --log-level warn --log-prefix "TCP FLAGS = FIN-RST: "
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags FIN,RST FIN,RST -j DROP
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags FIN,ACK FIN -j LOG --log-level warn \
    --log-prefix "TCP FLAGS = FIN no ACK: "
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags FIN,ACK FIN -j DROP
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags PSH,ACK PSH -j LOG --log-level warn \
    --log-prefix "TCP FLAGS = PSH no ACK: "
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags PSH,ACK PSH -j DROP
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags URG,ACK URG -j LOG --log-level warn \
    --log-prefix "TCP FLAGS = URG no ACK: "
$IPTABLES -A TCP_FLAG -p tcp --tcp-flags URG,ACK URG -j DROP
```

```

####
#
# SOURCE_CHECK chain
#
# Checks for invalid (reserved/spoofed) source address
#
# Note: this could be streamlined somewhat with better masking in some areas
# (e.g. /3 instead of /8); that might be somewhat more efficient, but for now
# we're listing each Class A address block individually to make the script
# easier to read
#
# See http://www.iana.org/assignments/ipv4-address-space for up-to-date
# listing of assigned addresses.

$IPTABLES -A SOURCE_CHECK -s 1.0.0.0/8 -j LOG --log-level warn \
    --log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 1.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 2.0.0.0/8 -j LOG --log-level warn \
    --log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 2.0.0.0/8 -j DROP

##
#
# NOTE: In reality, we also include the 5.0.0.0/8 block; for purposes of this
# text, however, we comment it out since this is the address we're pretending
# to use. Note that in reality, this is of course not the address block GIAC
# Enterprises uses!
#
### $IPTABLES -A SOURCE_CHECK -s 5.0.0.0/8 -j LOG --log-level warn \
###    --log-prefix "SPOOFSOURCE: "
### $IPTABLES -A SOURCE_CHECK -s 5.0.0.0/8 -j DROP
#

$IPTABLES -A SOURCE_CHECK -s 7.0.0.0/8 -j LOG --log-level warn \
    --log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 7.0.0.0/8 -j DROP

#
# We leave this next address block for the "EX_SOURCE_CHECK" so we don't drop
# all of our own traffic!
#
### $IPTABLES -A SOURCE_CHECK -s 10.0.0.0/8 -j LOG --log-level warn \
###    --log-prefix "SPOOFSOURCE: "
### $IPTABLES -A SOURCE_CHECK -s 10.0.0.0/8 -j DROP
#

$IPTABLES -A SOURCE_CHECK -s 23.0.0.0/8 -j LOG --log-level warn \
    --log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 23.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 27.0.0.0/8 -j LOG --log-level warn \
    --log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 27.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 31.0.0.0/8 -j LOG --log-level warn \
    --log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 31.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 36.0.0.0/8 -j LOG --log-level warn \

```

```

--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 36.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 37.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 37.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 39.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 39.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 41.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 41.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 42.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 42.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 49.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 49.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 50.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 50.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 58.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 58.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 59.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 59.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 60.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 60.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 69.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 69.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 70.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 70.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 71.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 71.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 72.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 72.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 73.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 73.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 74.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 74.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 75.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 75.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 76.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 76.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 77.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 77.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 78.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "

```



```

$IPTABLES -A SOURCE_CHECK -s 78.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 79.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 79.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 82.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 82.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 83.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 83.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 84.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 84.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 85.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 85.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 86.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 86.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 87.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 87.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 88.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 88.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 89.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 89.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 90.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 90.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 91.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 91.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 92.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 92.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 93.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 93.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 94.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 94.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 95.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 95.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 96.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 96.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 97.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 97.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 98.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 98.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 99.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 99.0.0.0/8 -j DROP

```

```

$IPTABLES -A SOURCE_CHECK -s 100.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 100.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 101.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 101.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 102.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 102.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 103.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 103.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 104.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 104.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 105.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 105.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 106.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 106.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 107.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 107.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 108.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 108.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 109.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 109.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 110.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 110.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 111.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 111.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 112.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 112.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 113.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 113.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 114.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 114.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 115.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 115.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 116.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 116.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 117.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 117.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 118.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 118.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 119.0.0.0/8 -j LOG --log-level warn \

```

```

--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 119.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 120.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 120.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 121.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 121.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 122.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 122.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 123.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 123.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 124.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 124.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 125.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 125.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 126.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 126.0.0.0/8 -j DROP

#
# Don't log or drop legitimate loopback traffic!
#
$IPTABLES -A SOURCE_CHECK -s 127.0.0.0/8 -i ! lo -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 127.0.0.0/8 -i ! lo -j DROP

$IPTABLES -A SOURCE_CHECK -s 197.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 197.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 221.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 221.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 222.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 222.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 223.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 223.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 240.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 240.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 241.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 241.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 242.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 242.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 243.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 243.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 244.0.0.0/8 -j LOG --log-level warn \
--log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 244.0.0.0/8 -j DROP

```

```

$IPTABLES -A SOURCE_CHECK -s 245.0.0.0/8 -j LOG --log-level warn \
    --log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 245.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 246.0.0.0/8 -j LOG --log-level warn \
    --log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 246.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 247.0.0.0/8 -j LOG --log-level warn \
    --log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 247.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 248.0.0.0/8 -j LOG --log-level warn \
    --log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 248.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 249.0.0.0/8 -j LOG --log-level warn \
    --log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 249.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 250.0.0.0/8 -j LOG --log-level warn \
    --log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 250.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 251.0.0.0/8 -j LOG --log-level warn \
    --log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 251.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 252.0.0.0/8 -j LOG --log-level warn \
    --log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 252.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 253.0.0.0/8 -j LOG --log-level warn \
    --log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 253.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 254.0.0.0/8 -j LOG --log-level warn \
    --log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 254.0.0.0/8 -j DROP
$IPTABLES -A SOURCE_CHECK -s 255.0.0.0/8 -j LOG --log-level warn \
    --log-prefix "SPOOFSOURCE: "
$IPTABLES -A SOURCE_CHECK -s 255.0.0.0/8 -j DROP

```

```

####
#
# EX_SOURCE_CHECK chain
#
# Check to make sure packets from the outside aren't spoofing our internal
# address
#
$IPTABLES -A EX_SOURCE_CHECK -i $EXT_IF -s 10.0.0.0/8 -j LOG --log-level warn \
    --log-prefix "SPOOFSOURCE: "
$IPTABLES -A EX_SOURCE_CHECK -i $EXT_IF -s 10.0.0.0/8 -j DROP

```

```

####
#
# TCP chain - filters all tcp traffic (IP protocol 6)
#
# We allow outbound Web (& DNS) traffic. We will also allow SSH and PPTP.
# Normal Web surfing traffic, and associated replies

$IPTABLES -A TCP -i $INT_IF -o $EXT_IF -p tcp --dport 80 -m state --state NEW \
    -j ACCEPT
$IPTABLES -A TCP -i $INT_IF -o $EXT_IF -p tcp --dport 443 -m state --state NEW \
    -j ACCEPT
$IPTABLES -A TCP -i $EXT_IF -o $INT_IF -p tcp --sport 80 -m state --state \

```

```

        ESTABLISHED,RELATED        -j ACCEPT
$IPTABLES -A TCP -i $EXT_IF -o $INT_IF -p tcp --sport 443 -m state --state \
        ESTABLISHED,RELATED        -j ACCEPT

# Since we're allowing outbound PPTP...
# Outbound PPTP
$IPTABLES -A TCP -i $INT_IF -o $EXT_IF -p tcp --dport 1723 -m state \
        --state NEW -j ACCEPT
$IPTABLES -A TCP -i $EXT_IF -o $INT_IF -p tcp --sport 1723 -m state --state \
        ESTABLISHED,RELATED        -j ACCEPT

# SSH-2, for maintenance & log checking
$IPTABLES -A TCP -i $EXT_IF -s $GIAC_SNAT_IP -p tcp --dport 22 -m state \
        --state NEW -j ACCEPT
$IPTABLES -A TCP -o $EXT_IF -d $GIAC_SNAT_IP -p tcp --sport 22 -m state \
        --state ESTABLISHED,RELATED -j ACCEPT

```

```

####
#
# UDP chain – filters all udp traffic (IP protocol 17)
#
# Obviously, we allow only the UDP we want, according to our security policy
# -- which is outbound DNS queries, plus we also want to allow our firewall to
# use an outside NTP server.
#
# We wish we could provide L2TP over IPSec capability as well, but we're using
# NAT so that won't work.

# First, allow DNS queries. (As with Firewall A, we're only allowing udp,
# since we don't want zone transfers. This isn't compliant with the DNS
# specifications; large replies will try to use TCP and fail. That shouldn't be
# a problem for us, though; if anything, blocking such large DNS replies may
# help block would-be cache poisoning. See RFC 1035, which describes DNS, at
# http://www.ietf.org/rfc/rfc1035.txt

# DNS
$IPTABLES -A UDP -i $INT_IF -o $EXT_IF -p udp --dport 53 -m state --state NEW \
        -j ACCEPT
$IPTABLES -A UDP -i $EXT_IF -o $INT_IF -p udp --sport 53 -m state --state \
        ESTABLISHED,RELATED        -j ACCEPT

# NTP, so these logs are reasonably close to our other machines' logs
$IPTABLES -A UDP -o $EXT_IF -s $PUB_FW -d $PUB_NTP1 -p udp \
        --dport 123 -m state --state NEW -j ACCEPT
$IPTABLES -A UDP -o $EXT_IF -s $PUB_FW -d $PUB_NTP2 -p udp \
        --dport 123 -m state --state NEW -j ACCEPT
$IPTABLES -A UDP -i $EXT_IF -s $PUB_NTP1 -d $PUB_FW -p udp \
        --sport 123 -m state --state NEW -j ACCEPT
$IPTABLES -A UDP -i $EXT_IF -s $PUB_NTP2 -d $PUB_FW -p udp \
        --sport 123 -m state --state NEW -j ACCEPT

# Log & drop anything else from the outside, it's unwanted
$IPTABLES -A UDP -i $EXT_IF -p udp -j LOG --log-level warn \
        --log-prefix "DISALLOWED UDP: "
$IPTABLES -A UDP -i $EXT_IF -p udp -j DROP

```

```
#####
#
# ICMP chain – filters all icmp traffic (IP protocol 1)
#
# We only allow the stuff we really need
#

# We want to be able to PING
$IPTABLES -A ICMP -o $EXT_IF -s $INT_NET -p icmp --icmp-type 8 -m state \
    --state NEW -j ACCEPT
$IPTABLES -A ICMP -i $EXT_IF -d $INT_NET -p icmp --icmp-type 0 -m state \
    --state ESTABLISHED,RELATED -j ACCEPT

#
# Destination Unreachables, so we don't have to wait to time out
#
$IPTABLES -A ICMP -i $EXT_IF -d $INT_NET -p icmp --icmp-type 3 -m state \
    --state ESTABLISHED,RELATED -j ACCEPT
#
# Let the outside world think they're able to ping us
#
$IPTABLES -A ICMP -i $EXT_IF -p icmp --icmp-type 8 -m state \
    --state NEW -j REJECT --reject-with icmp-host-prohibited

# Log & Drop all the rest
$IPTABLES -A ICMP -p icmp -j LOG --log-level warn --log-prefix "DENIED ICMP: "
$IPTABLES -A ICMP -p icmp -j DROP
```

```
#####
#
# OTHER chain - this filters traffic of any other IP protocol
#
# We'll let our guests use GRE (type 47) in case they need PPTP.
#

# Allow GRE for PPTP
$IPTABLES -A OTHER -p 47 -i IN_IF -o $EXT_IF -j ACCEPT
$IPTABLES -A OTHER -p 47 -i EXT_IF -o $INT_IF -j ACCEPT

#
# Drop any other type of traffic
#
$IPTABLES -A OTHER -j LOG --log-level warn --log-prefix "ODD PROTOCOL: "
$IPTABLES -A OTHER -j DROP
```

```
#####
#
# OUTPUT chain – this filters traffic generated by the firewall itself
#
# Since we're checking our logs manually on this firewall, the only thing
# outbound should be NTP requests and SSH replies. Those are, of course,
# handled in the appropriate protocol chains.
#
```

```
# Send everything to the appropriate chain
$IPTABLES -A OUTPUT -p tcp -j TCP
$IPTABLES -A OUTPUT -p udp -j UDP
$IPTABLES -A OUTPUT -p icmp -j ICMP
$IPTABLES -A OUTPUT -j OTHER

# Log & drop everything that gets returned to here...

$IPTABLES -A OUTPUT -j LOG --log-level warn --log-prefix "FW COMPROMISE?: "
$IPTABLES -A OUTPUT -j DROP
```

```
#####
#
# POSTROUTING chain -- this just Source NAT's our outbound traffic, since our
# private addresses aren't routable and we need a valid public address to
# communicate with the outside world. We do no filtering here, that was done in
# the "filter" table already; now we're in the "nat" table, "it should only be
# used to translate the packet's source field or destination field"18

# We SNAT everything headed to the Internet (which is why we can't provide IPsec
# for our visitors).
$IPTABLES -t nat -A POSTROUTING -i $INT_IF -o $EXT_IF -j SNAT \
    --to-source $SNAT_IP
```

```
# This concludes our ruleset creation!
```

```
####
#
# Kernel preparations, now that Netfilter is up & running.
#
# Note that these could also be put into a different startup script entirely
#
```

```
touch /var/lock/subsys/local
# Make sure the firewall responds to NAT replies
/sbin/ifconfig eth0:0 5.6.7.14 netmask 255.255.255.240

# Drop ICMP broadcasts in kernel
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
```

```
# Drop source-routed packets in kernel
for f in /proc/sys/net/ipv4/conf/*/accept_source_route; do
    echo 0 > $f
done
```

```
# Drop ICMP redirect packets in kernel
for f in /proc/sys/net/ipv4/conf/*/accept_redirects; do
    echo 0 > $f
done
```

```
# Do not create ICMP redirect packets in kernel
for f in /proc/sys/net/ipv4/conf/*/send_redirects; do
    echo 0 > $f
```

---

<sup>18</sup> Oskar Andreasson, <http://www.netfilter.org/documentation/tutorials/blueflux/iptables-tutorial.html#AEN595>

done

```
# Just in case this didn't happen elsewhere already,  
# enable IP forwarding so we can route  
echo 1 > /proc/sys/net/ipv4/ip_forward
```

---

As with our primary Firewall A, we will make a bootable CD for actual implementation of this firewall.

Now that we've got 2 nice, new firewalls up and running, we can breathe a little sigh of relief -- though we certainly have no false beliefs that we're safe or secure by any means. Firewalls and packet filtering in general are only one part of an effective security policy. They are certainly a very important part, but monitoring and maintenance are just as critical. This is not a case of "set it and forget it."

We also can't rest for too long because we're not done building our perimeter yet. We still need to configure our remote access.

### 3. VPN(s)

GIAC Enterprises will have 3 methods of remote access; each of the three could be called a "Virtual Private Network." Those three methods are Citrix NFuse (in conjunction with Citrix Secure Gateway), SSH-2, and L2TP over IPsec. The overwhelming majority of our remote access will take place using Citrix NFuse, because it provides very fast response, even over dial-up modems; it can be accessed from any contemporary browser with an Internet connection; and, it is secure.<sup>19</sup> SSH is provided for IT staff, in the event they need to monitor or manage something remotely. The L2TP over IPsec connection is provided for our "road warriors," staff outside of the office who need to synchronize Microsoft Outlook for off-line use.

Because these off-site staff are often using slow, dial-up modem connections to get to the Internet, they will first use the Citrix NFuse Website to check the status of their e-mail. If they find new messages they need access to offline, or have time-sensitive e-mail they composed offline and need to send out, they will then connect to the L2TP over IPsec VPN in order to synchronize their Outlook clients. This is generally viewed as a method of last resort, due to the large amount of bandwidth involved; synchronizing Outlook via dial-up has been compared to having teeth pulled while watching grass grow.

---

<sup>19</sup> [http://www.rsasecurity.com/company/news/releases/pr.asp?doc\\_id=1264](http://www.rsasecurity.com/company/news/releases/pr.asp?doc_id=1264), "RSA Security and Citrix Systems Provide Stronger Security for the Virtual Workplace" and [http://www.rsasecurity.com/company/news/releases/pr.asp?doc\\_id=180](http://www.rsasecurity.com/company/news/releases/pr.asp?doc_id=180), RSA Security Expands Licensing of its E-Security Software to Citrix



For similar reasons, they won't perform Windows 2000 off-line files synchronization remotely. They will instead transfer files via the Citrix connection on an as-needed basis.

We will discuss here the process of setting up the L2TP over IPsec server. For Citrix NFuse and SSH-2, we will discuss the important specifics but will spare you the step-by-step configuration. Citrix NFuse configuration, in particular, could easily comprise a book by itself.

### **A. Windows 2000 Routing & Remote Access (RRAS) using L2TP over IPsec (a mini-tutorial).**

We will be keeping the existing RRAS server (minus the dial-in modems), but we want to harden this server and tighten its security as much as possible. PPTP access was previously used, but this has known security limitations (basically, the quality of the encryption hinges upon the complexity of the user's password; see <http://www.counterpane.com/pptpv2-paper.html>, "Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2)"). Instead, we will configure the server to allow only L2TP with IPsec access for roaming staff. We use both the RRAS server's filtering capabilities and those of Firewall A to restrict, as much as we can, what the clients who connect here will be able to reach. While for the time being we are only implementing the server for client access, our configuration will make it possible to create server to server VPN connections using L2TP with IPsec with a minimal amount of effort; all that would be required would be to create the connection (our side of it, anyway) and modify our filter sets as needed, depending on which servers access was allowed to. [Though how easy or hard this would really be is of course contingent upon the type of hardware & software used on the other end; if they aren't using Windows 2000's RRAS, all bets are off...]

We are able to use IPsec, even though our internal network uses NAT, because we aren't altering the tunneled packets. We intentionally put our L2TP server on a separate firewall interface and gave it a "real" public IP address. While the traffic to and from this interface will be filtered by the firewall, the packets we allow will be forwarded without alteration. The VPN server will then give the connected clients ip addresses on the NAT'ed network; since that traffic is all encapsulated within the IPsec traffic, NAT isn't broken. This is best explained with a diagram, shown in

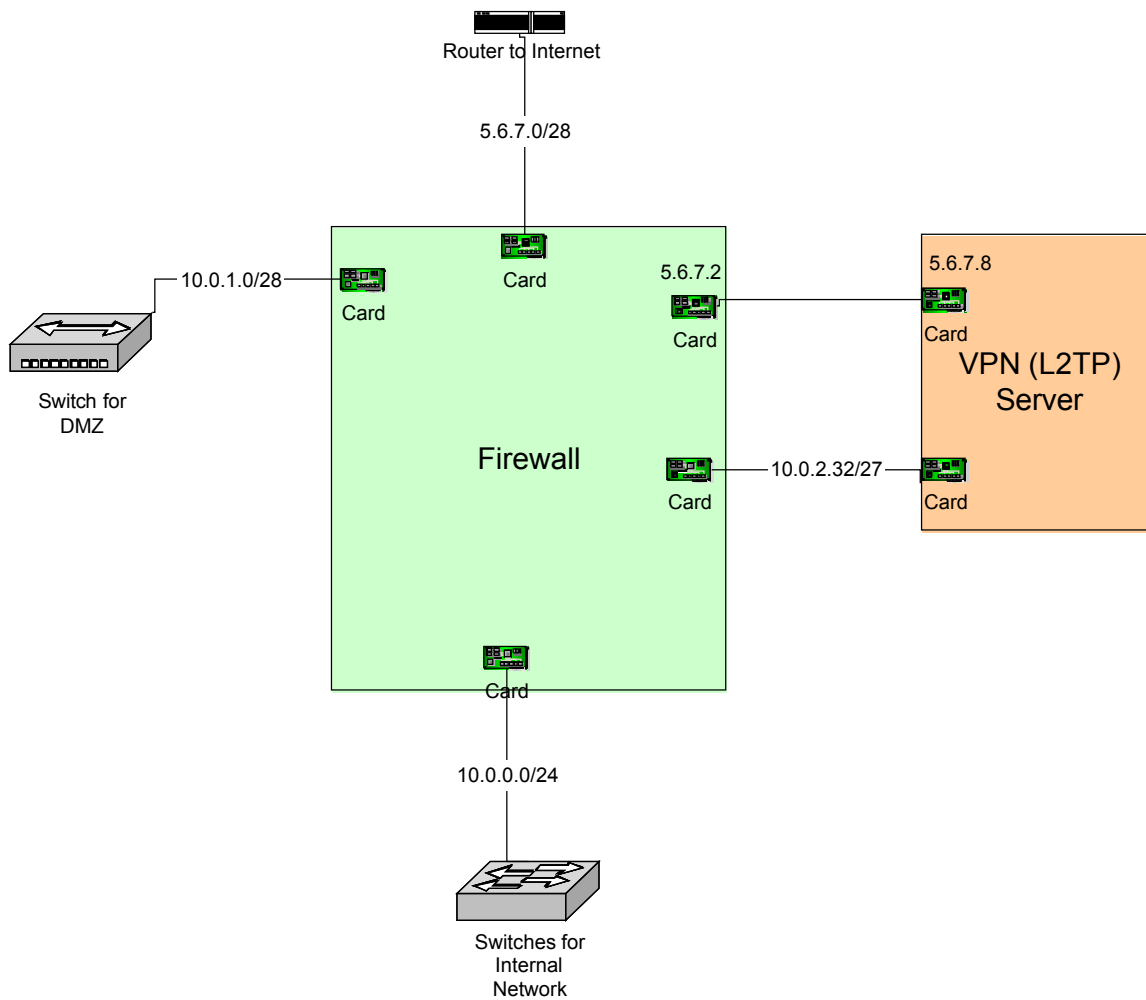
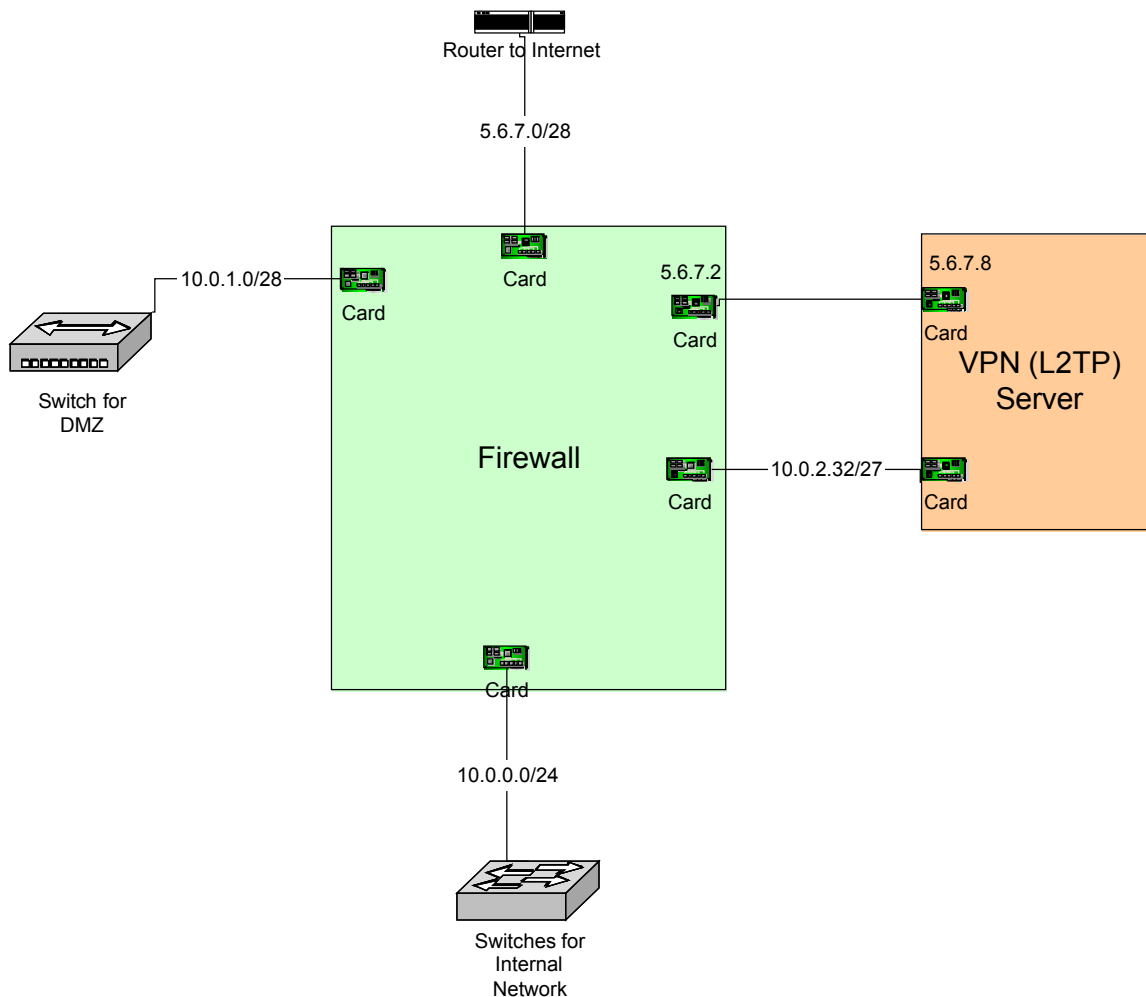


Figure 24 below.

© SANS Institute



**Figure 24**

As there was no way to ensure our existing server hadn't already been compromised, it was decided after careful deliberation that the best course of action was to simply rebuild the machine from scratch. [This course of action was also taken with the other existing servers, including the Web servers running IIS and the SQL server. Whenever possible, a new machine was configured ahead of time, which was then swapped with the existing server in a matter of minutes.]

Again, the details of hardening a Windows 2000 computer are beyond the scope of this document. We will, however, go over the basics. First, before the computer is physically connected to the network, the operating system (Windows 2000 Server) is installed, followed immediately by all relevant patches and updates. These patches and updates are installed from CD, rather than over the network, to prevent the machine from being compromised before we can patch it. Note that during installation, no unnecessary services, such as IIS, or unnecessary programs, such as the "accessories," were installed.

Out Secure Servers,” also by <http://www.technet.microsoft.com/ArticleID=24892>.



**Figure 25, Disabling services through the MMC**

Next, we install and configure the personal firewall software and antivirus software. Although these can be distributed and managed remotely, for the servers in the DMZ we'll install these programs from CD, so we can have them fully functional before we ever connect to the network. Because we're dealing with a very limited number of machines, this does not present a problem. For our L2TP server, we need to open the following traffic inbound from the Internet side interface:

1. UDP destination port 500, for IKE
2. UDP destination port 1701, for L2TP
3. IP Protocol 50 (ESP) for IPSec

We also need to allow the outbound reply traffic (i.e., source ports of 500 and 1701).

The internal interface is much more complicated. This is because we need to allow the traffic required to authenticate both the VPN server itself and the remote users on the Active Directory domain, and we need the ability to communicate with the Exchange 2000 server. We have 3 domain controllers, and we'll enable authentication traffic to all three.

This requires opening the following traffic inbound from the internal side of the L2TP VPN server to the Global Catalogue Domain Controller and the Exchange server on the internal network. These ports are the same we configured in our primary firewall A's ruleset above.

1. TCP, UDP 88 - These are used by Kerberos
2. TCP 135 - RPC portmapper; this falls in the category of "ports you have a firewall to block access to." Opening the firewall to this port is definitely a concern, but our risk is minimized because we're authenticating by both computer certificate and username & password to get to this point.
3. UDP 137 & 138 - NetBIOS & login; thought it would work without it, but it didn't...
4. TCP 139 - NetBIOS, login
5. TCP 389 - LDAP
6. TCP 445 - Netlogon; this, too, is definitely something to block normally.

The default list of ports required by the Exchange server could be summed up as, "Pretty much everything." However, we'll make some registry changes to slim this down a little bit. First, we'll follow the recommendations in Microsoft KB article Q270836, "XCLN: Exchange 2000 Static Port Mappings." On the Exchange server – after a good backup, of course! – we will add a value to the HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\MSExchangeSA\Parameters subkey. We create a new value called TCP/IP

Port, Data type: REG\_DWORD. The Radix is Decimal; for Value data, we enter the port number we want to use. It needs to be between 1024 and 5000; we will use 4444, simply because it's easy to remember. (Note that this port is actually assigned to a couple of different services -- but since we don't use either of them, this doesn't present a problem.) In this same subkey, we also need to add a REG\_DWORD value called TCP/IP NSPI Port for the Directory NSPI Proxy Interface; we will assign it a decimal value of 4445. Lastly, we also need to add a TCP/IP Port value on the Exchange 2000 server to the HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\MSExchangeIS\Parameters\System subkey. For this Information Store parameter, we'll use port 4446.

We also need to specify a range of RPC Service ports. Normally, Windows 2000 simply uses dynamically allocated ports from anywhere in the upper range (1024-65,535). Opening up all of those is a bigger risk than we feel comfortable with, so we'll follow the guidelines in Microsoft KB article Q154596, "HOWTO: Configure RPC Dynamic Port Allocation to Work with Firewall." This leads us to the following Registry changes on our Exchange 2000 server and our Domain Controllers. Under the HKEY\_LOCAL\_MACHINE\Software\Microsoft\Rpc key, we add a new key called Internet. We then create a new value of type REG\_MULTI\_SZ called Ports. (Note that to creating this type of value requires using REGEDT32.EXE rather than the older REGEDIT.EXE!) The data we put in will be a range, 5555-5655. We can really use just about any range we want; Microsoft suggests using ports above 5000, and allocating a minimum of 20. We've chosen numbers that are easy to remember, and keep us way above the minimum; since we've got lots of internal clients connecting as well, we'd rather not take the risk of running low. Basically, we're balancing the needs for security with the needs for stability.

Within this same Internet key, we also need to create two more values, both of type REG\_SZ. The first of these is called PortsInternetAvailable, the second is UseInternetPorts. These values take case-insensitive data values of Y or N; the PortsInternetAvailable value specifies whether our range entered in the Ports value is the ports available to use (Y) or the ports excluded from use (N). As with the rest of our security policies, we're specifying only what is allowed, so we assign a value of Y. The UseInternetPorts value specifies whether or not to limit our ports to those specified in the Ports value -- basically, this value tells our server to implement the changes we've just made. We naturally give it a value of Y.

Note that if Exchange were running on the Global Catalogue server itself, there'd be another registry change; see KB article Q298369, "How to Configure a Global Catalog Server to Use a Specific Port When Servicing MAPI Clients."

Now that we've made all of these registry changes, we say a quick prayer and reboot our Exchange server to implement them. (Tip: if you stop the Exchange

services manually before rebooting, you can save yourself as much as 45 minutes on this step!)

We'll have to enable all of the ports mentioned above in our VPN filters, and on our primary firewall as well (between the NIC on the L2TP server's inside adapter and the firewall's inside adapter; this was already taken care of in the firewall script shown above.)

As a result, we need to allow the following additional traffic out the interface on the internal side:

4. TCP ports 4444, 4445 and 4446 to the Exchange server
5. TCP 5555-5655 - RPC Service ports, as per our registry hack.
6. TCP 3268 - Global Catalogue access to the GC domain controller

We also need to enable traffic to our DNS and NTP servers:

7. UDP 53 - DNS, so our clients know where to find our GC and Exchange servers. (We could, if we chose, use manual entries in the Hosts files of the clients to avoid needing to open this up. For the time being, we have chosen to simply use DNS to avoid the administrative headache.)

8. UDP 514 - Syslog; this is so that our VPN server's NTSyslog software can send the logs to our syslog server. (Note that this is opened to the DMZ syslog server, rather than the internal server. Why allow unneeded traffic to the internal network?)

9. UDP 123 - NTP; this is so the time on our VPN server can synchronize.

We will also need to allow the replies associated with all of this traffic.

We are now ready to connect our machine to the network, and join it to the GIACENT.COM Active Directory domain. We make it part of the domain because we will be using Windows to authenticate our connecting users.

Before we configure the Routing and Remote Access service, we will rename our network interfaces to make them easier to work with. This is shown below in *Figure 26*. This will make our lives much easier later on.

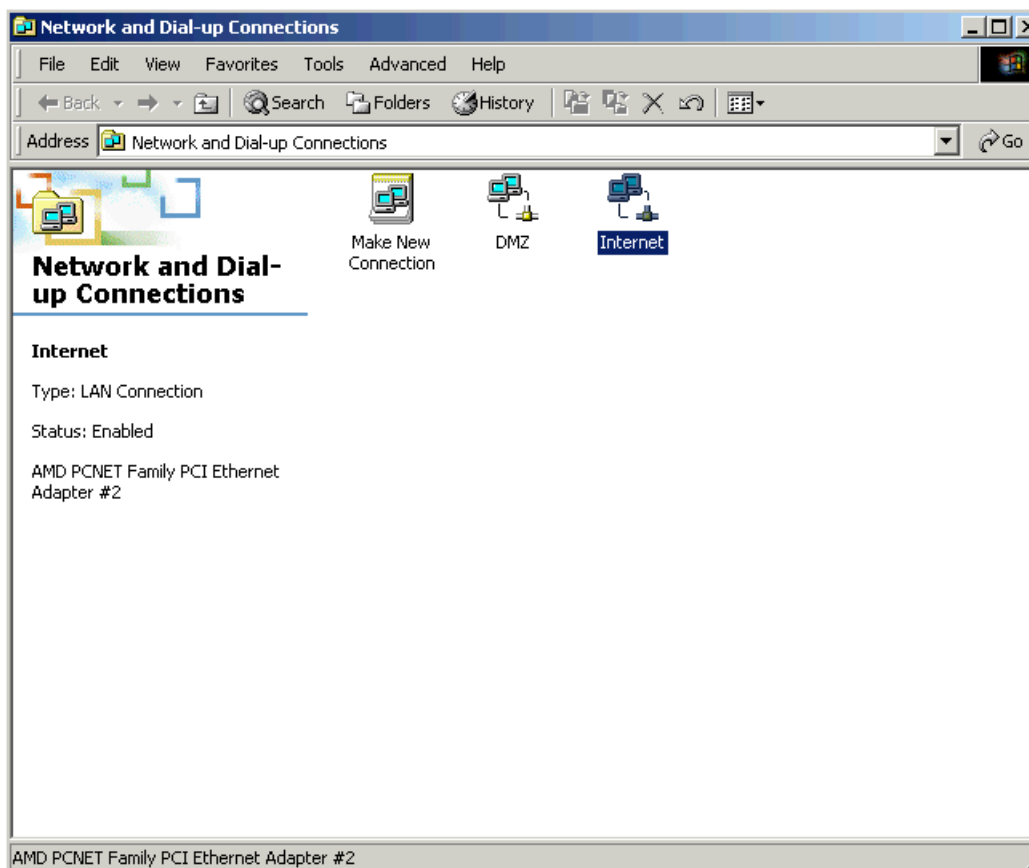


Figure 26

We are now finally ready to start configuring the Routing and Remote Access Service. This is done via the aptly named Routing and Remote Access MMC Snap-in, found under Start>Programs>Administrative Tools. We begin by right clicking on our server and selecting “Configure and Enable Routing and Remote Access.” Naturally, this brings up the welcome screen to the Routing and Remote Access Server Setup Wizard which basically tells us, in case we didn’t know, that we just clicked on “Configure and Enable Routing and Remote Access.” (As an aside, I wish we could turn off a lot of Windows 2000’s Wizards. At the very least, I’d like to be able to turn off these welcome screens.)

In *Figure 27* we see the first choice we have in the Wizard. We want to configure this server as a VPN server.



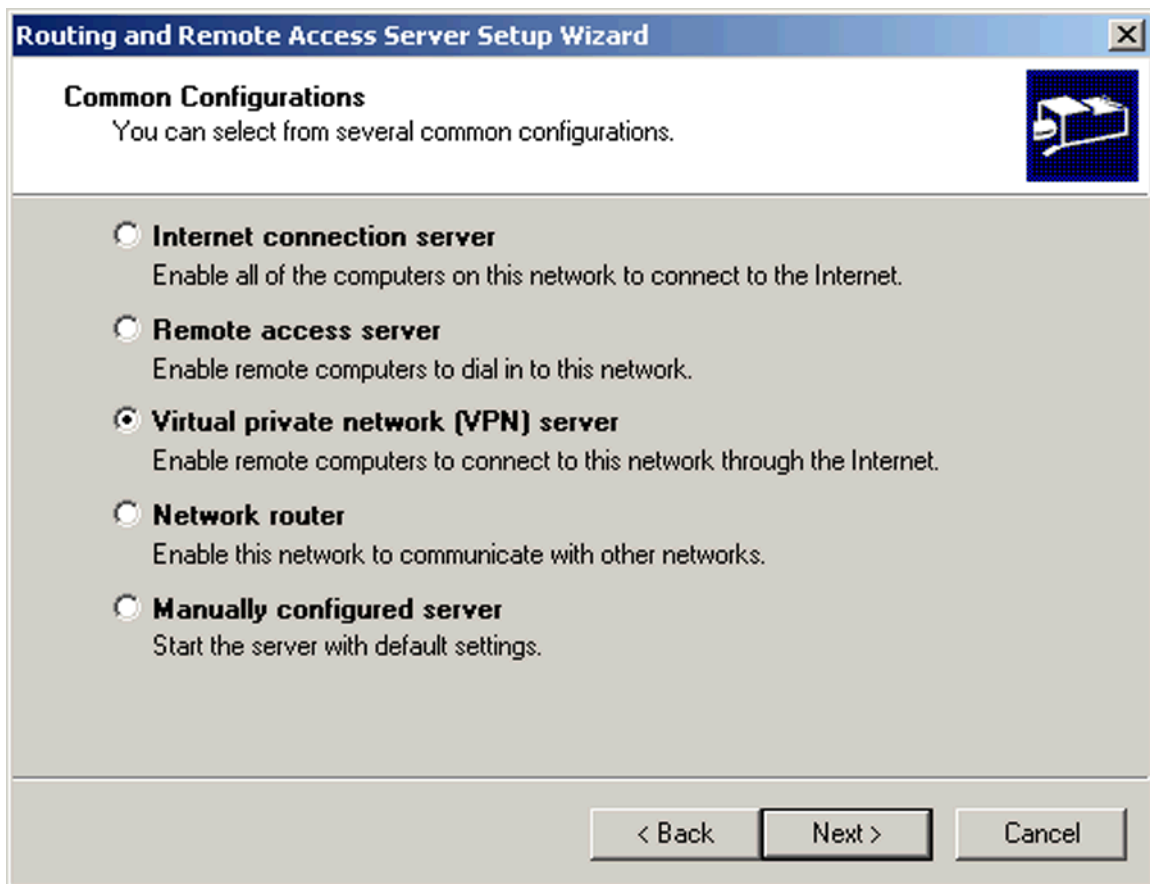


Figure 27

We are next asked to confirm we have the network protocols we need; since the only protocol we're using is TCP/IP, that's the only one shown. Note the wording of the Yes and No choices; we can't choose which protocols we want here, it's an all or nothing deal. Choosing "No" will exit the wizard, presumably to let you install the protocols you need. If we were also using something like IPX/SPX for a Novell network, we would have to enable that here also; we would then have to disable it later on.

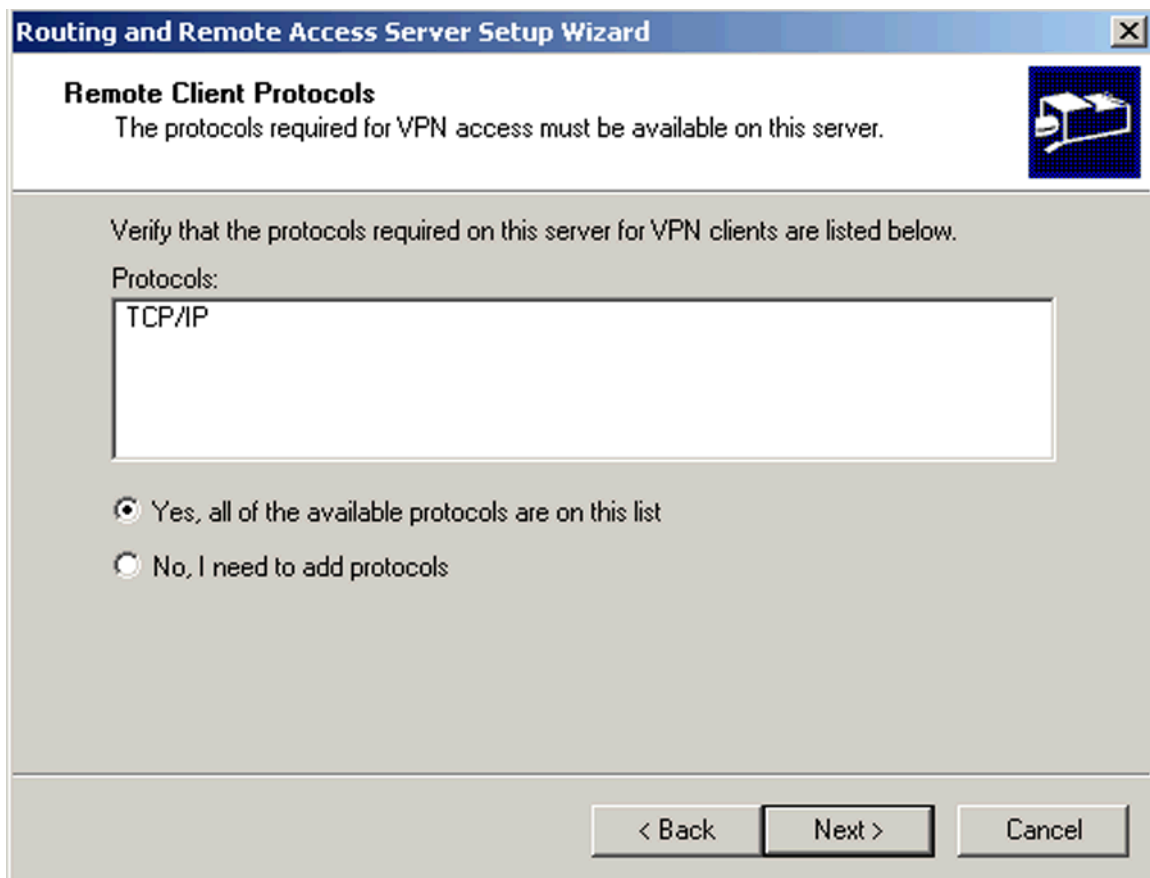


Figure 28

We are next asked to select which interface is on the Internet side, and which is on our protected network. This is one example where renaming the services earlier makes things a little less confusing later on; this is shown in *Figure 29*.

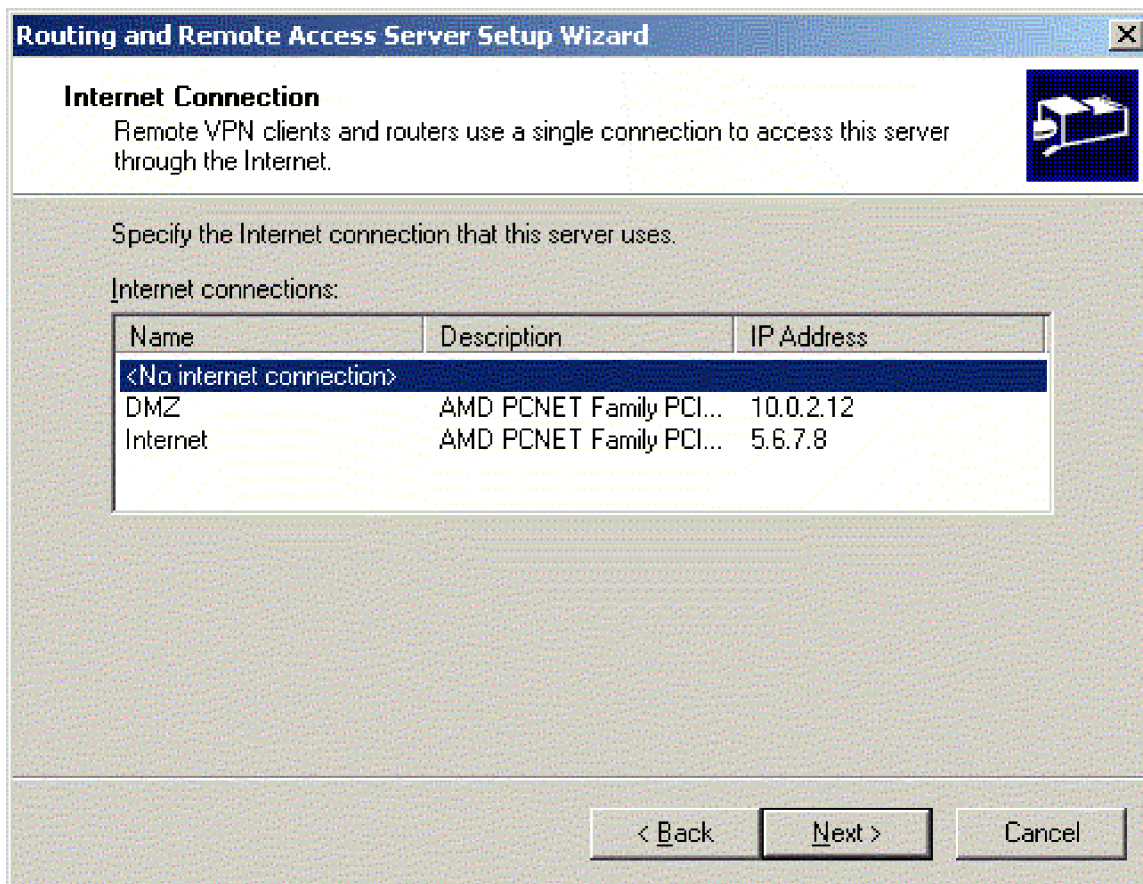


Figure 29

We are also asked which interface is on the LAN side.

We are then asked if we want to use DHCP to assign local IP addresses to our clients or assign them from a static pool. Because we know the range of IP addresses we want to use, and using DHCP would simply add complexity, we'll use addresses from a static pool. This is shown below in *Figure 30*.

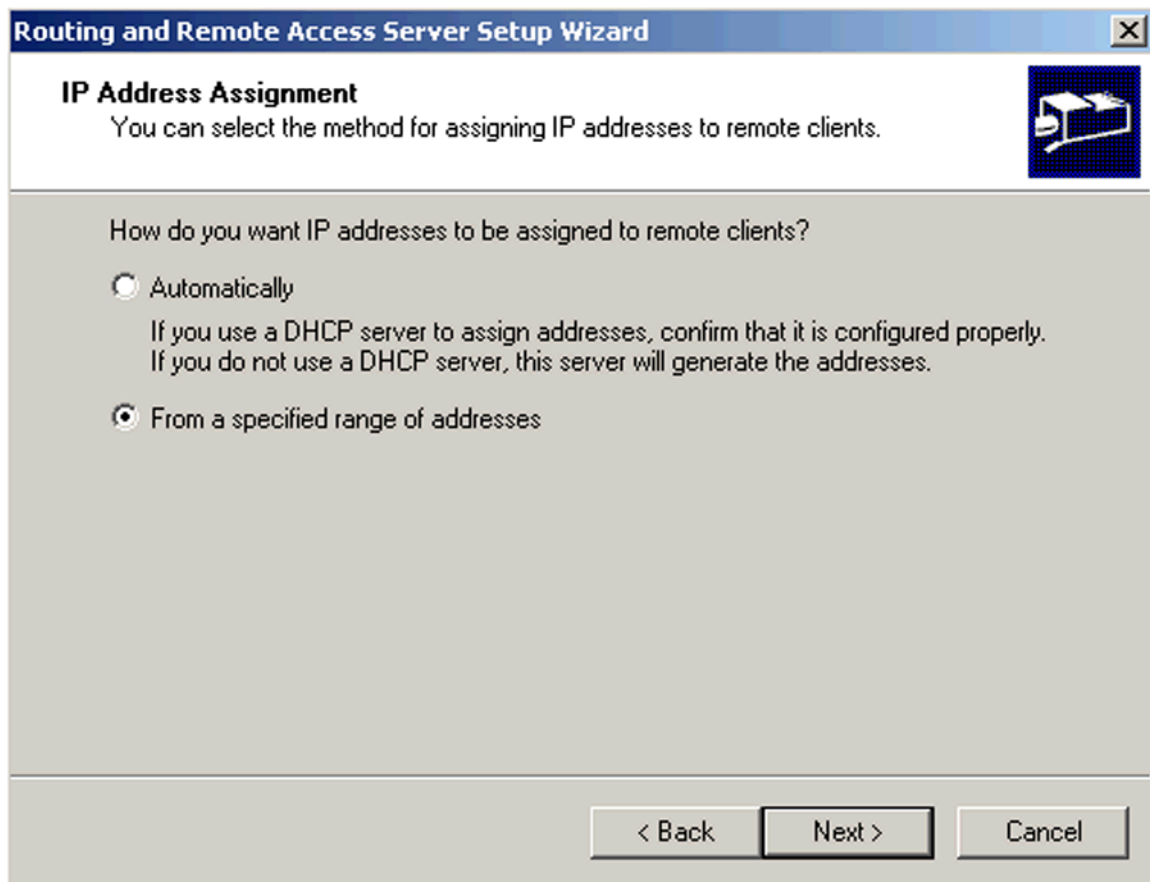


Figure 30

In Figure 31 we see our address range selection.

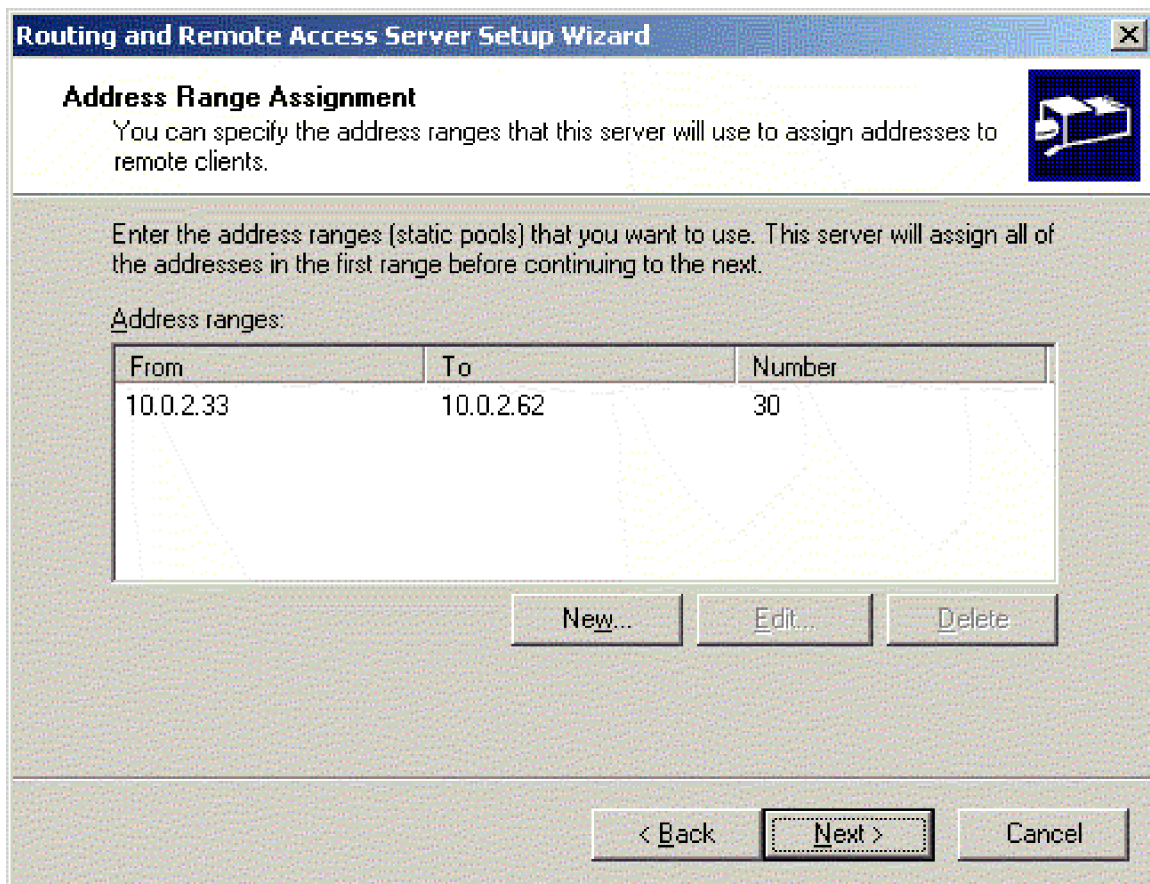


Figure 31

Next, we have to tell Windows whether or not we're using a RADIUS server for user authentication. Since we're a fairly small shop, don't have multiple VPN servers to synchronize, and are under severe pressure not to spend money, we're just going to use our existing Active Directory domain controllers for authentication. This does generate some cause for concern, as our domain controllers – by definition – contain all the keys to our little kingdom. This concern, in fact, is why our L2TP server is on a separate, dedicated network with both interfaces tied directly to dedicated interfaces on the primary firewall via cross-over cables. There is no way to "sniff" packets on this segment without compromising either the VPN server or the firewall itself; if either of those things were to happen, authenticating through an intervening RADIUS server that itself got its information from our domain controller would only provide a minimum amount of additional security anyway.

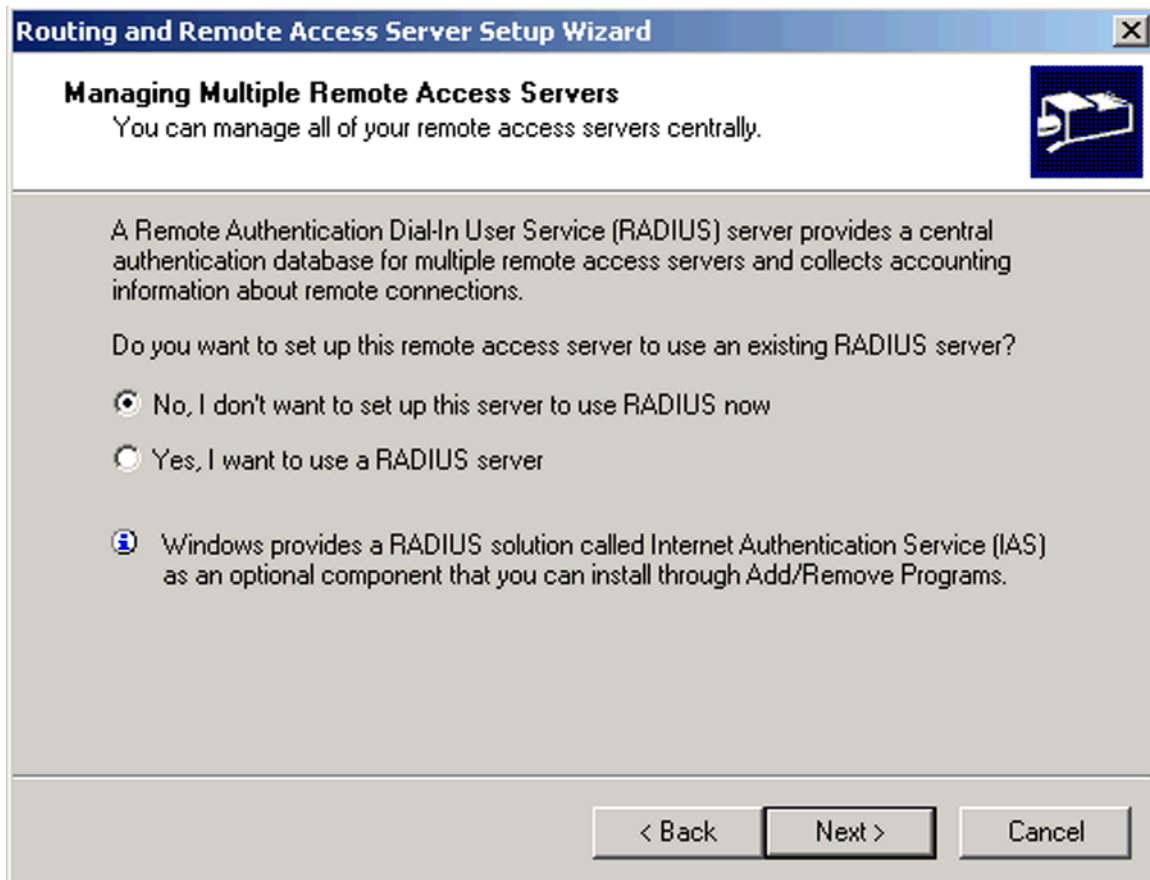
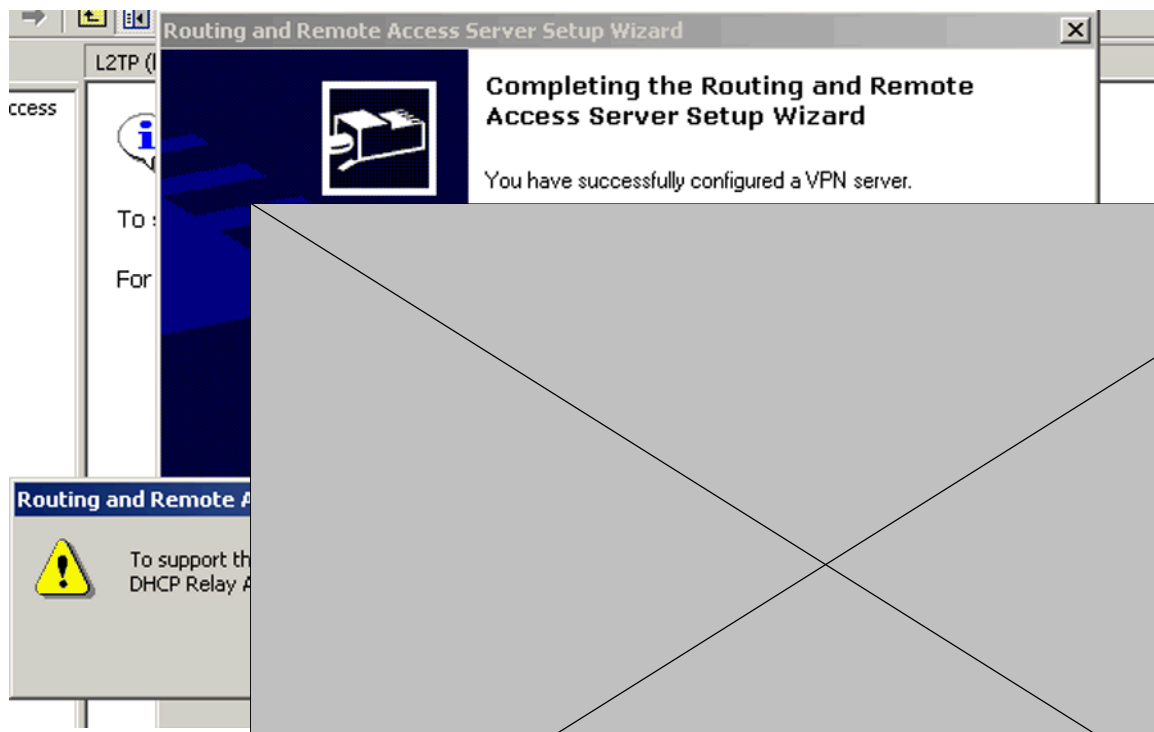


Figure 32

This ends our trip through the Routing and Remote Access Server Setup Wizard. We click Finish on the closing screen – and are, for no apparent reason, reminded that we haven't configured DHCP for our clients yet.



© SANS Institute 2000 - 2002,

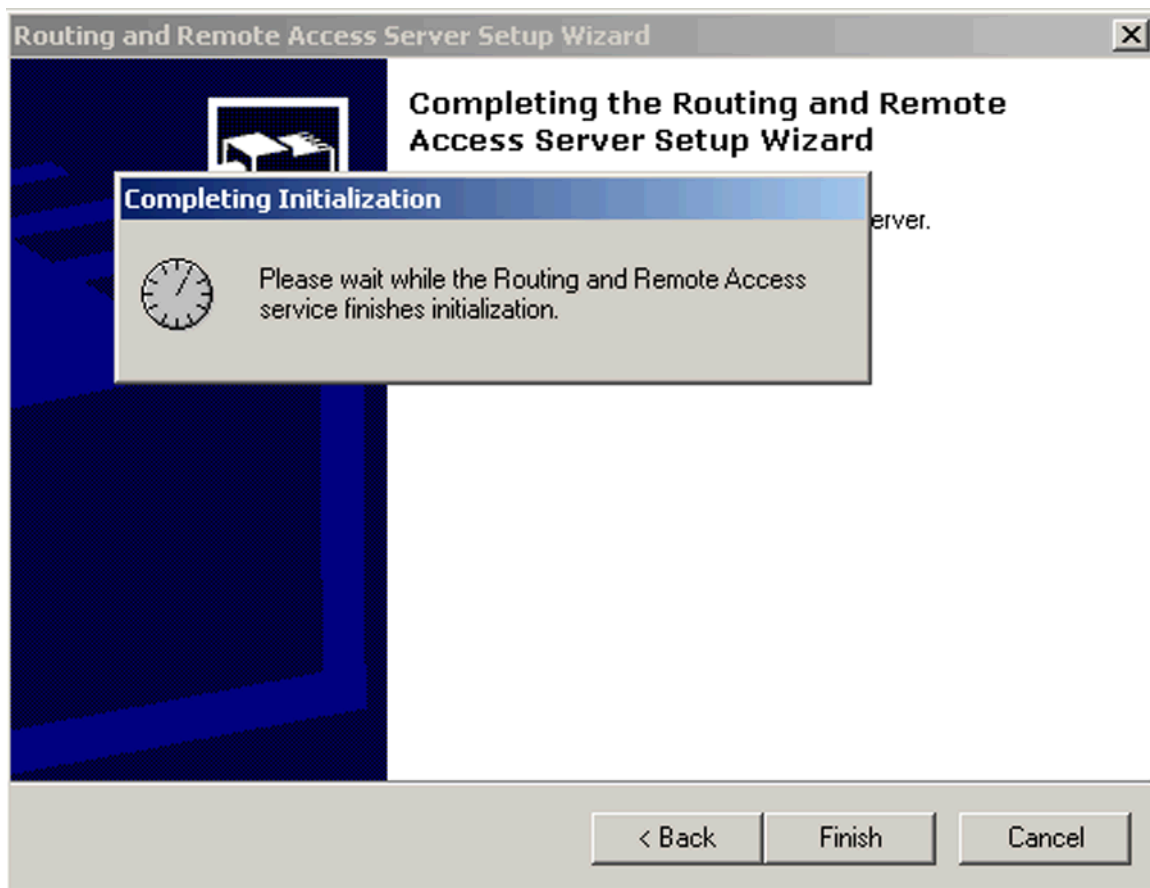


Figure 34

The first thing we notice is that the Wizard has generously configured our server for 256 concurrent connections (128 via PPTP and 128 via L2TP.) Rather curious, as we've already told it we've only got a range of 30 IP addresses.



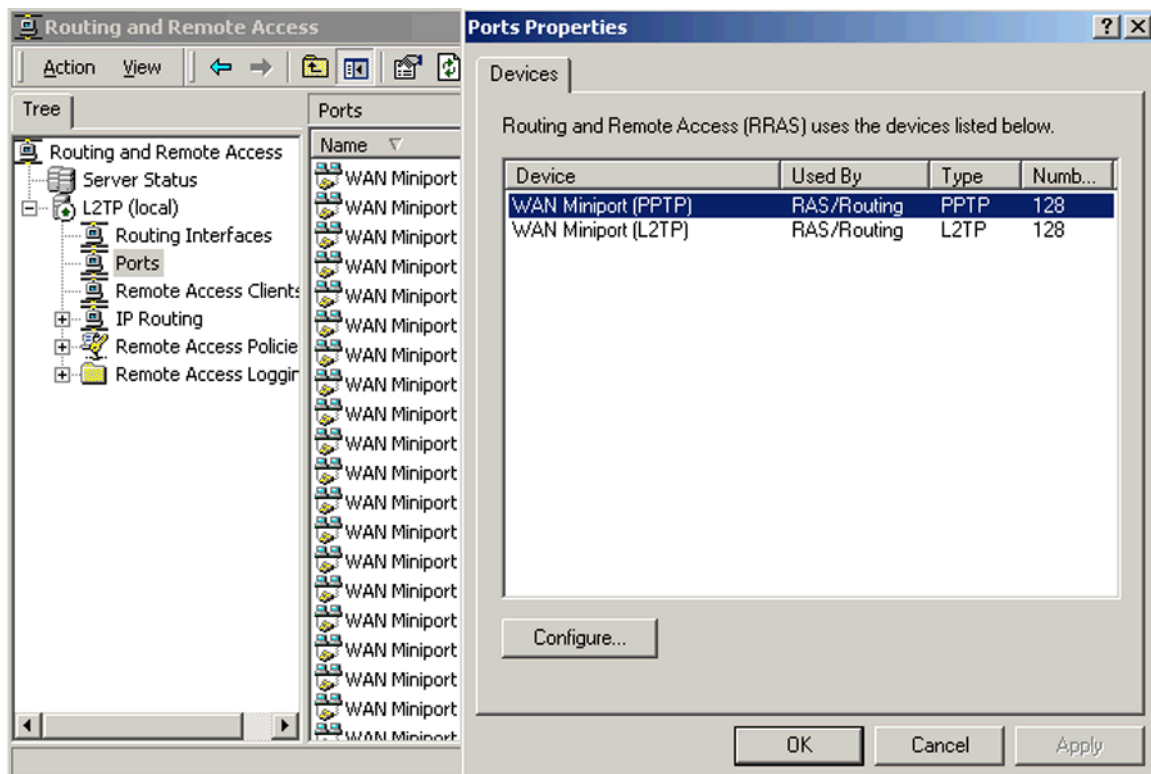


Figure 35

We'll fix this. First, we'll disable the insecure PPTP. Windows 2000 VPN clients will, by default, attempt to connect using L2TP; if that fails, they will then revert to PPTP. We want to ensure that if IPsec fails, the session fails. (Since our firewall doesn't allow GRE to the VPN server, PPTP would fail anyway.) Note in Figure 36 that Windows will not let us specify a number of PPTP ports lower than 1.

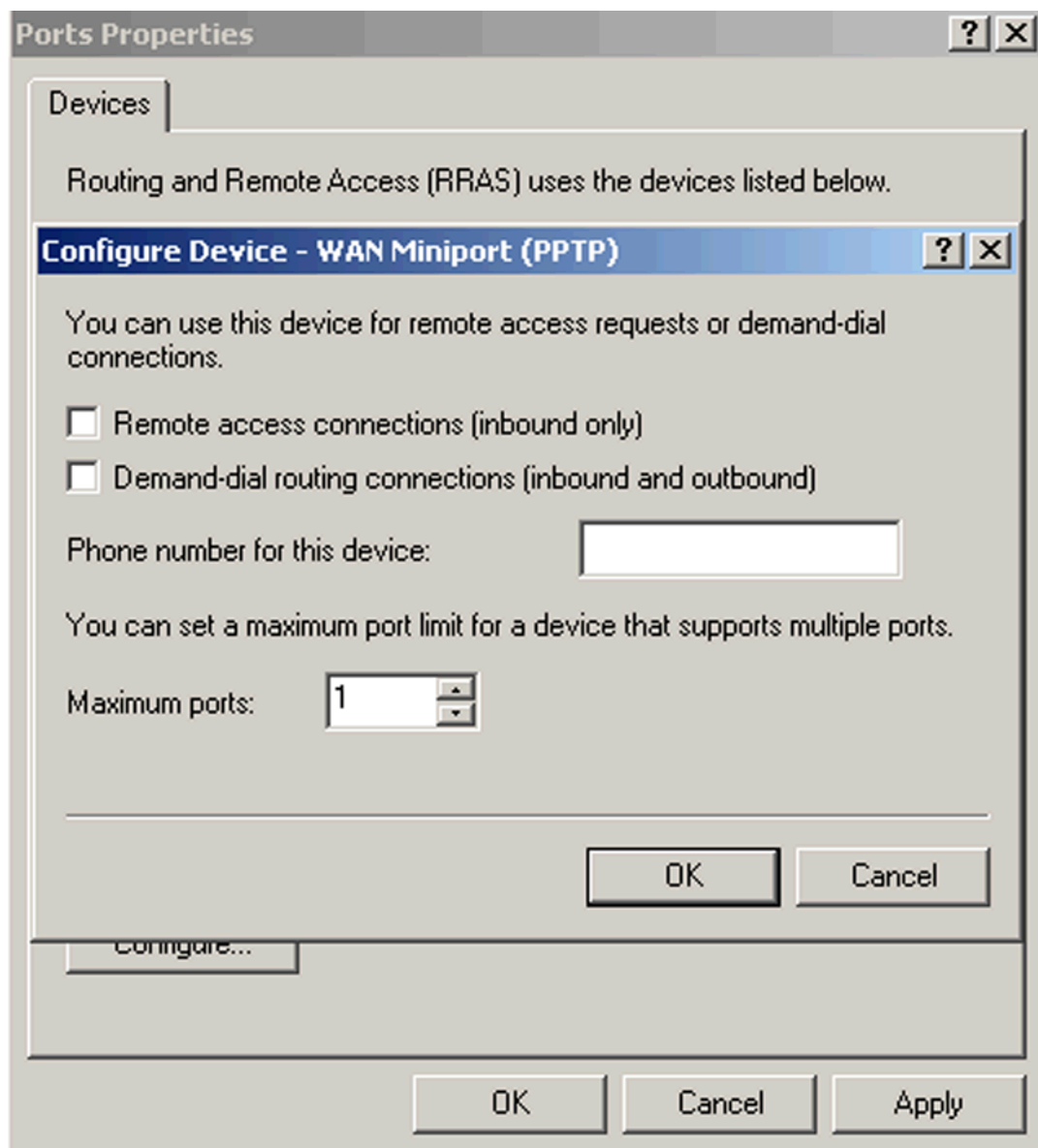


Figure 36

Next, we'll configure the L2TP ports to the number we want to allow. Note that we don't want our server to be able to generate sessions, only answer.

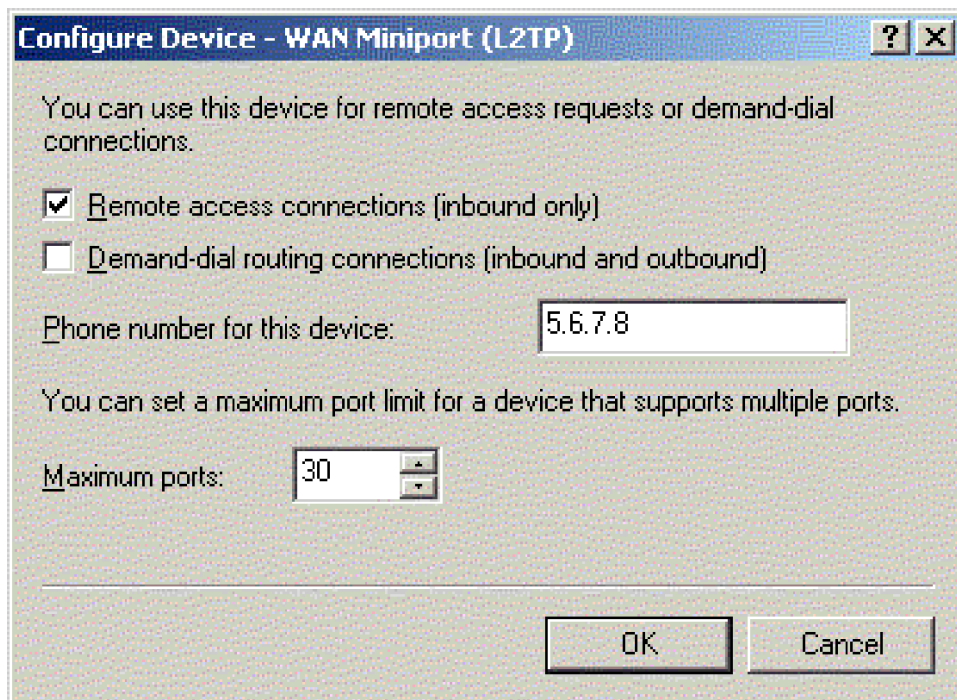


Figure 37

Next, we'll configure the properties of the RRAS server. We do this in the MMC by simply right-clicking on our server and selecting Properties. First, we see the General tab; we'll change this so it doesn't act as a router.

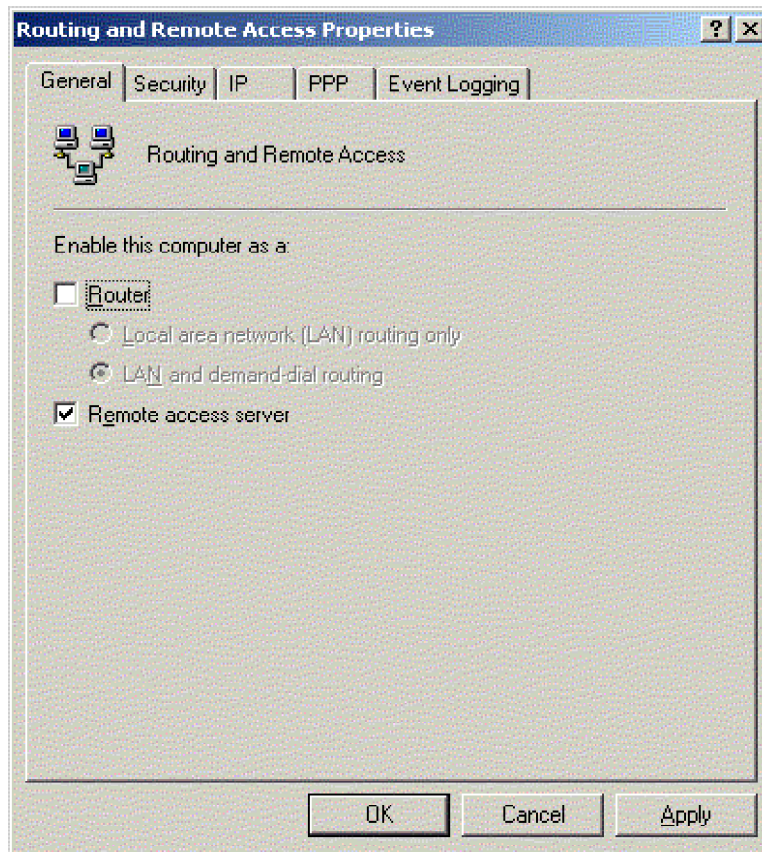


Figure 38

Next, we click on the Security tab, as shown in *Figure 39*. This is where we configure the authentication methods for L2TP. Note that this is somewhat confusing, as we're using IPsec and our sessions will use computer certificates to encrypt and ensure the IP communications. The settings here are for user authentication, after the IPsec tunnel has been established. We're using MS-CHAP v2; we'd use EAP if we had smart cards, but we don't. As mentioned earlier, MS-CHAP v2 does suffer from the fact that the strength of its encryption is contingent upon the complexity of the passwords; however, we are a) requiring complex passwords, and b) tunneling this over IPsec anyway, so this does not present a significant security problem for us. In other words, our clients must first authenticate via digital certificates for IPsec to function; they must then supply a valid username and password.

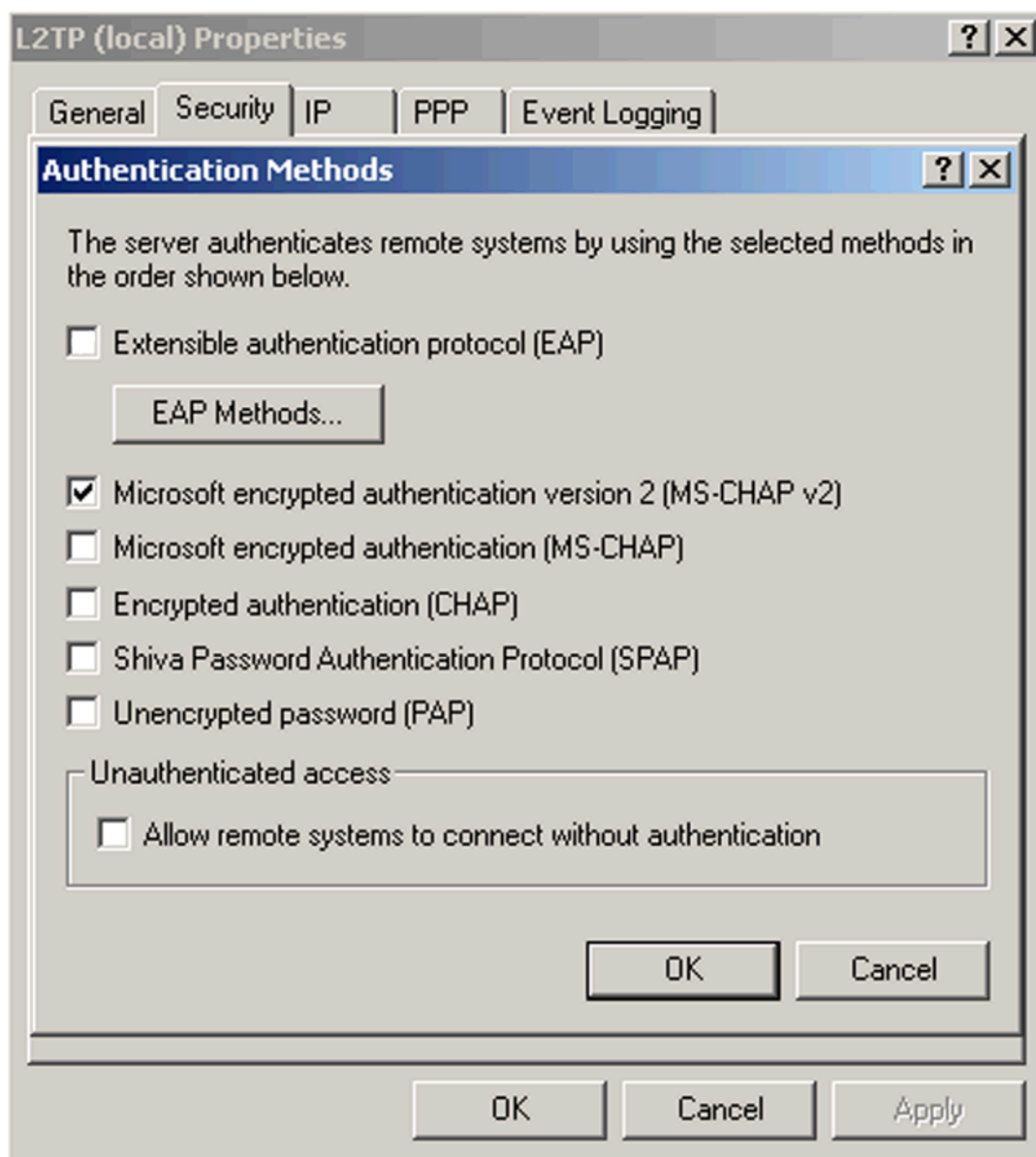
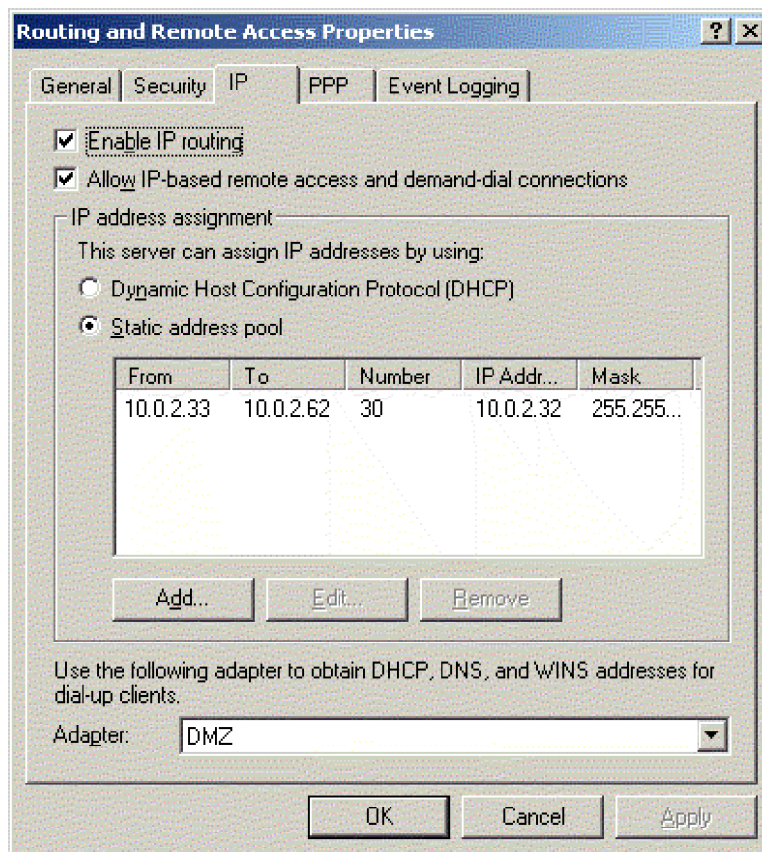


Figure 39

Figure 40 shows the IP tab. Because our ultimate goal is to access servers behind this one, we have to enable IP routing. While we're here, we'll double-check the IP addresses we'll be handing out, and set the adapter for clients to use to get networking services to use the internal DMZ adapter.



**Figure 40**

The “Event Logging” tab is where we tell the server how much, if at all, to log. We’re going for the maximum allowed here.

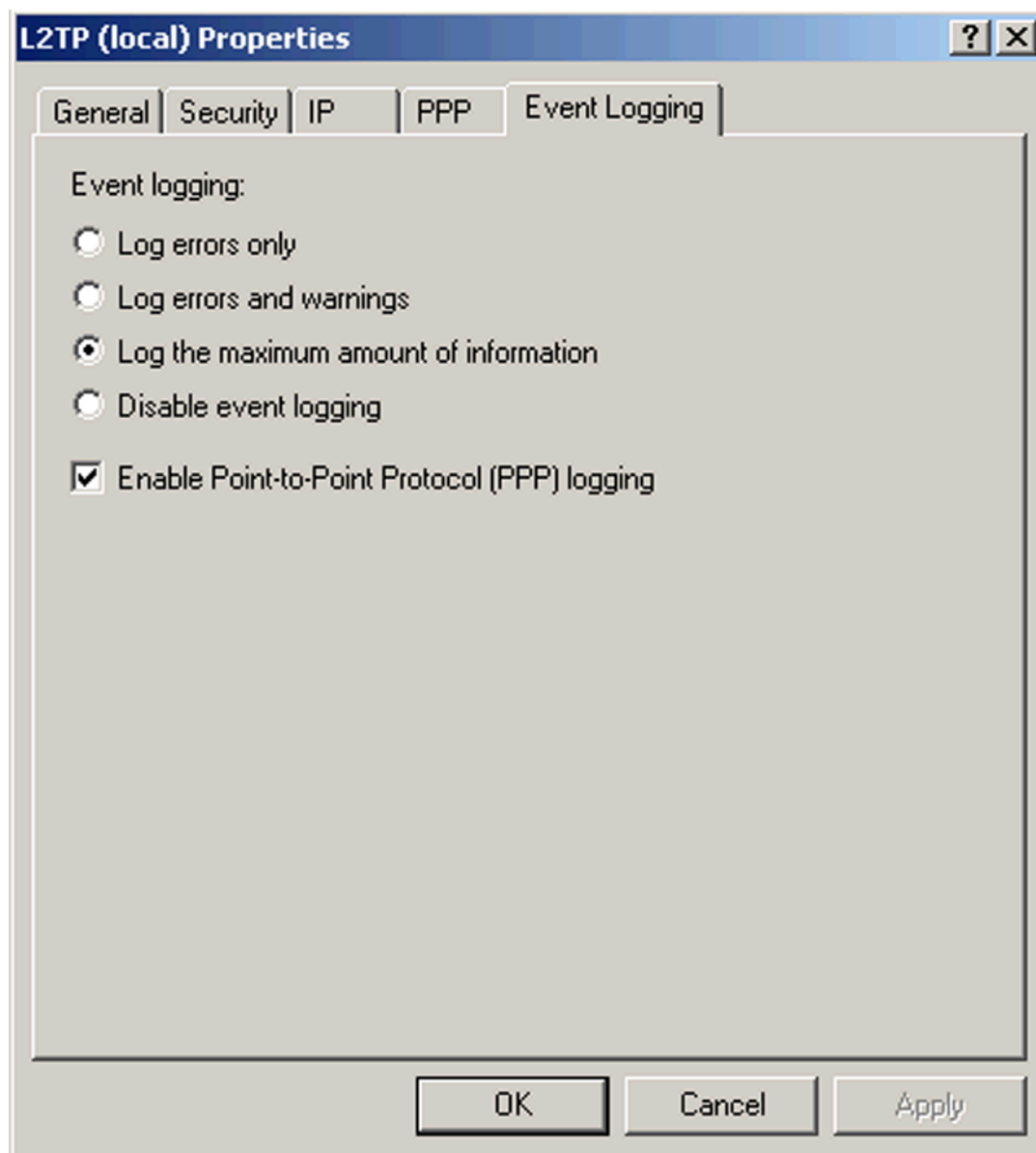
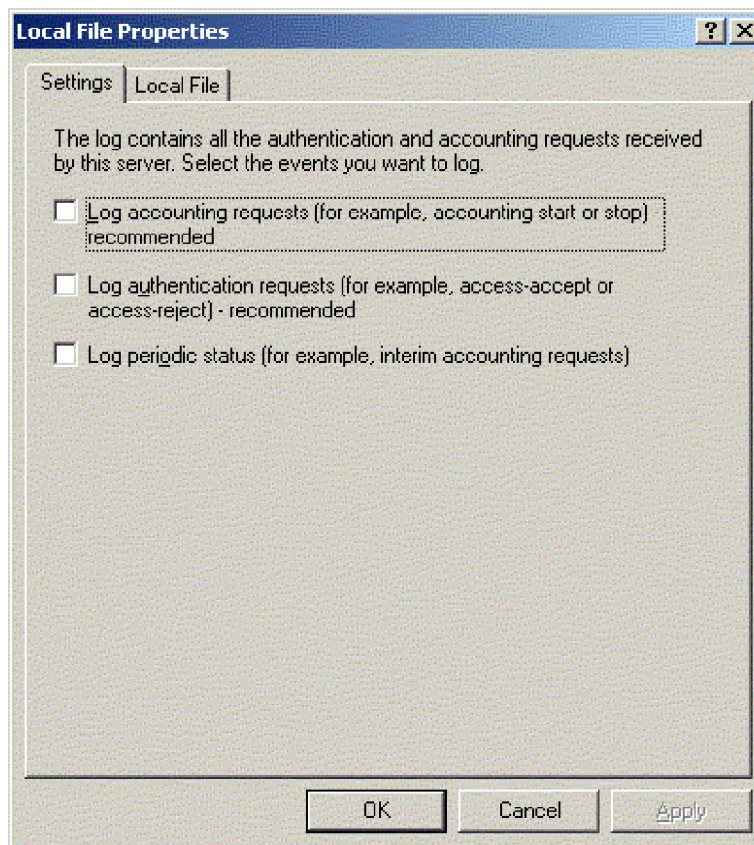


Figure 41

It's important to note, however, that this doesn't necessarily give us all the logging we could get. As we see in *Figure 42*, we still need to scroll down to "Remote Access Logging," right-click on the file, and check off the things we want our log file to include:



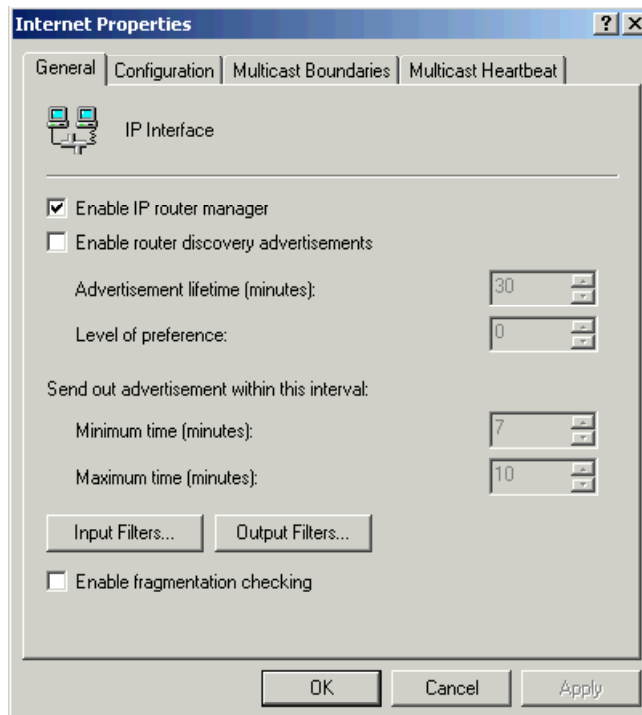


**Figure 42**

We now move on to configuring the filters for the interfaces. We start with the "Internet" external interface; all we want to allow is inbound UDP to ports 500 (IKE) and 1701 (L2TP), and outbound we allow traffic from those same ports. Note that we do not need to set filters here to allow IP protocol 50 (ESP), as the IPSec module takes affect, removing the headers, before (or after, depending on traffic direction) the packets reach these RRAS filters.

We get to the screens for configuring the filters by expanding the IP Routing node in the tree in the left-hand side of the MMC, highlighting General, and right-clicking the Internet interface in the resulting right-hand detail window and selecting Properties. This brings up the interface properties window shown in *Figure 43* below.





**Figure 43**

Obviously, we add input IP filters by clicking on the Input Filters button, and then clicking on the Add button. This brings up the screen shown in *Figure 44*, adding an IP filter in which we're adding the filter to allow traffic inbound to L2TP. Note that with Service Pack 3, we found filters pre-configured for us for UDP 1701 and 500.

The Output filters have the same numbers, but obviously in the opposite direction, with the source address of the VPN server's external interface.

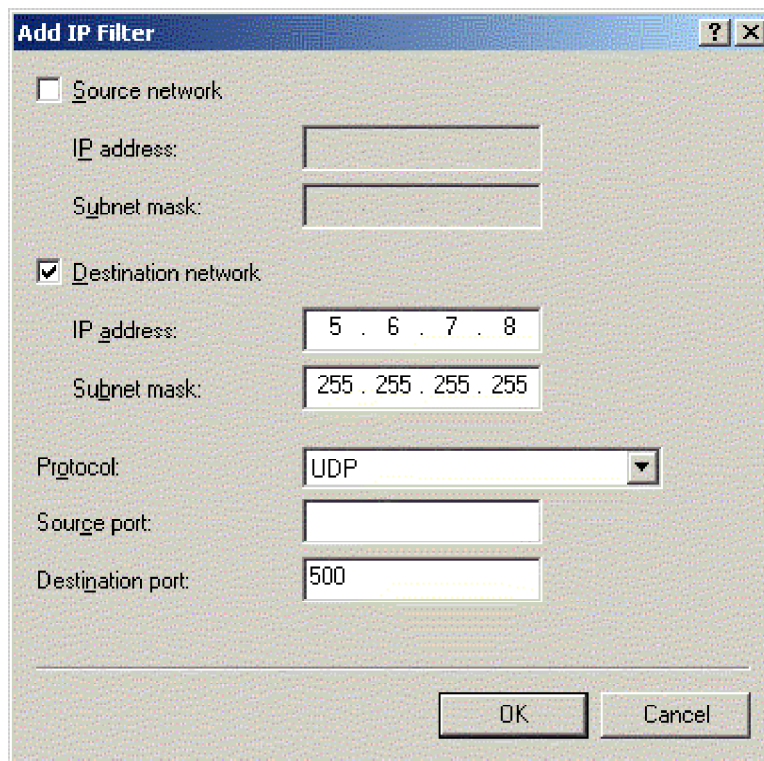


Figure 44, adding an IP filter

In *Figure 45* we have the completed Input filter for the Internet interface. Incoming traffic is only allowed to UDP ports 500 and 1701. Note also we have selected the radio button for "Drop all packets except those that meet the criteria below." This is in accordance with our overall policy of least required permissions; it is also *not* the default, which is to only reject packets that meet the filters defined here!

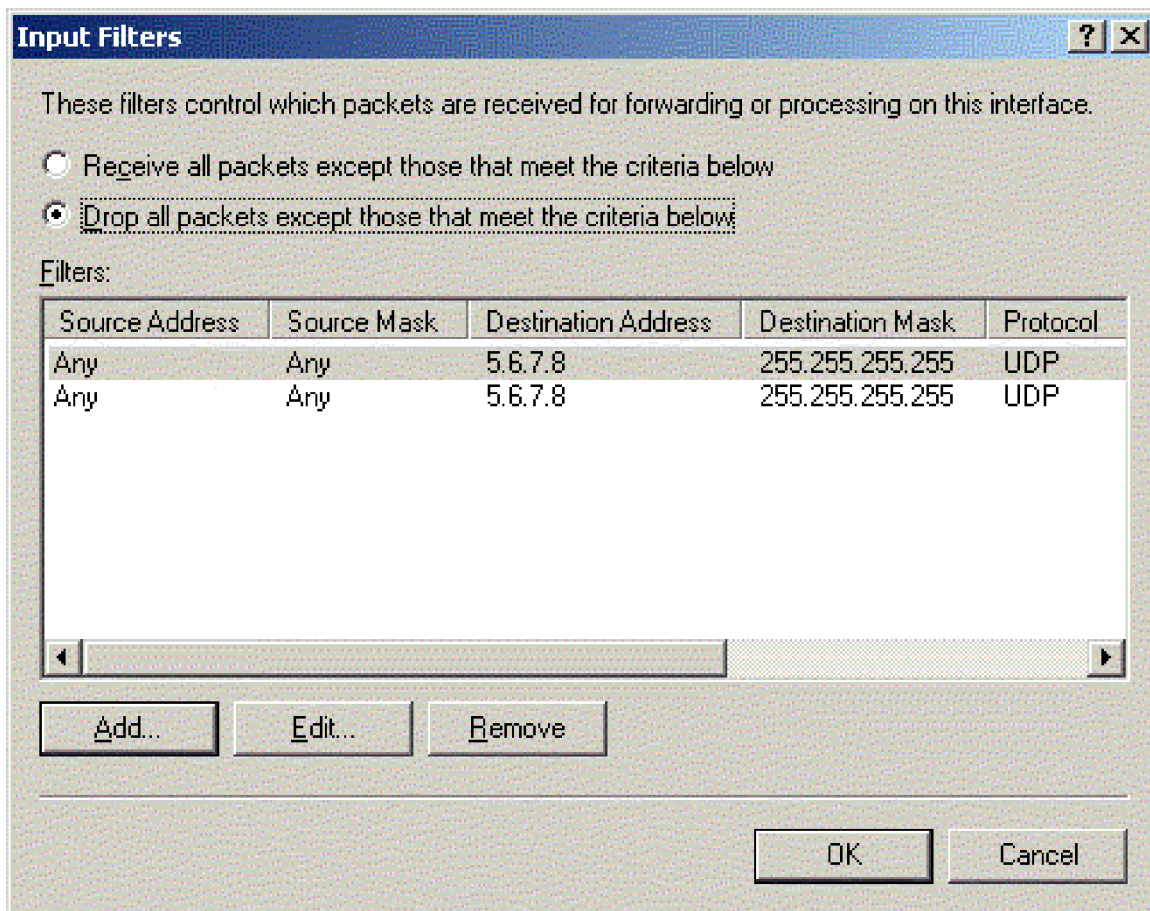


Figure 45

We will spare you the screenshots of configuring the internal interface. It is configured with an Input filter that is the same as those used in the personal firewall, outlined above. We allow only the required traffic to our Exchange and DC servers.

Now that we have our server configured, we need to configure the clients. The IPSec we're going to implement uses 3DES encryption. This is, of course, overkill for our purposes, but it's available so we'll use it. (That's easier than trying to explain to upper management why it's OK to use DES even though it's been cracked.) While we could theoretically use pre-shared keys, this is both more effort and less secure. Instead, we will use computer certificates issued by our Certificate Authority (a Windows 2000 server. Note that our "root" CA is an older machine kept off-line, only fired up when the issuing CA's certificate needs renewing.) This enables us to distribute our certificates via Active Directory's Group Policy, and revoke certificates easily if and when we need to. (We also use this CA to issue certificates to our partners for their authentication over the Web.)

Note that the clients will not get their certificates until Group Policy is refreshed.

This can be done via the command line but, given the technical prowess of our users, we simply tell them to reboot. It's important for them to do this while still connected to the internal network, as they won't be able to use the VPN connection to *get* their certificates before they *have* their certificates!

Actually configuring the client is pretty simple. We go to Start>Settings>Network and Dial-up Connections, and double-click on Make New Connection. Naturally, we first see the welcome screen telling us what we just clicked on; moreover, before we are given the option to click on Next, we are prompted with a Location Information window asking for information we must fill out "Before you can make any phone or modem connections..." My experience has been this window will pop up even on machines without a modem; but, clicking Cancel here will actually exit us out of the Network Connection Wizard. So, we simply enter our area code, click OK twice, and proceed on our way. Naturally, that means clicking Next on the welcome screen.

This brings up the screen we see in *Figure 46*, which asks us to choose the type of connection we want to make.

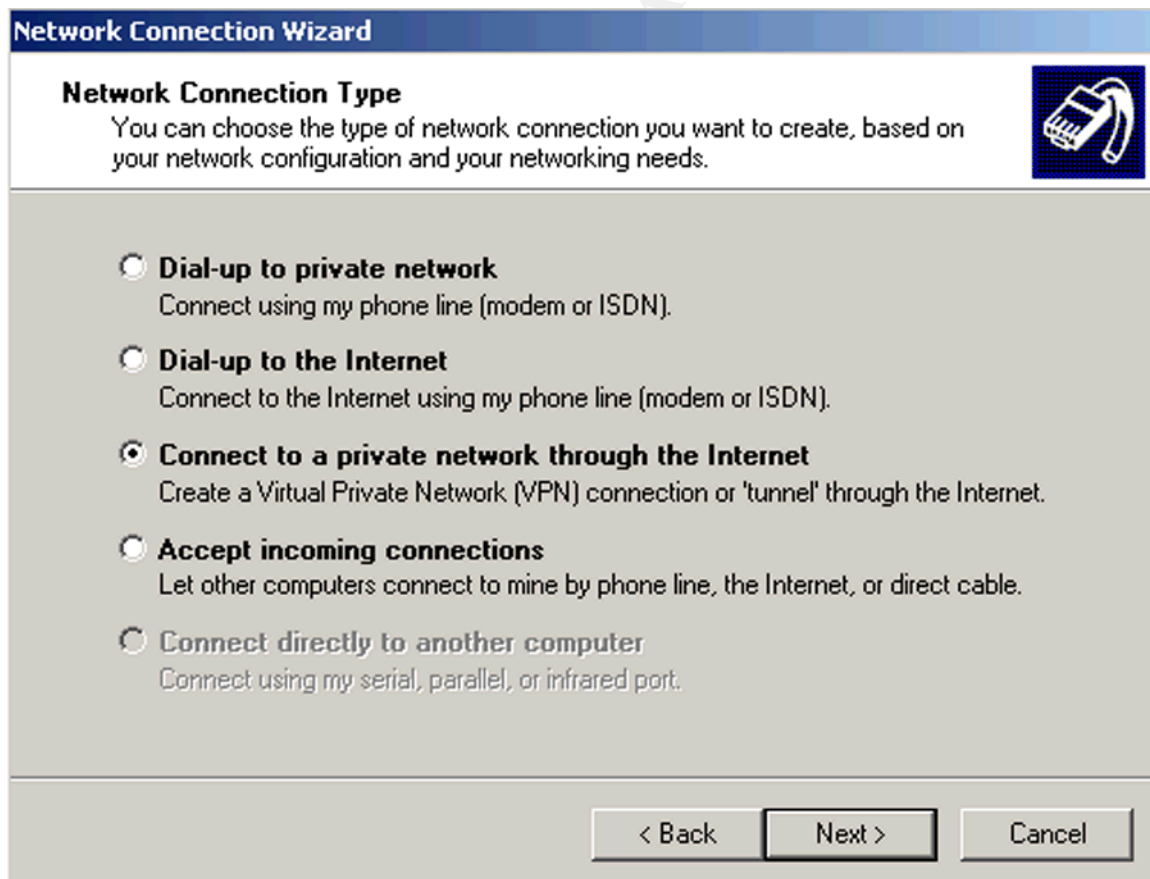
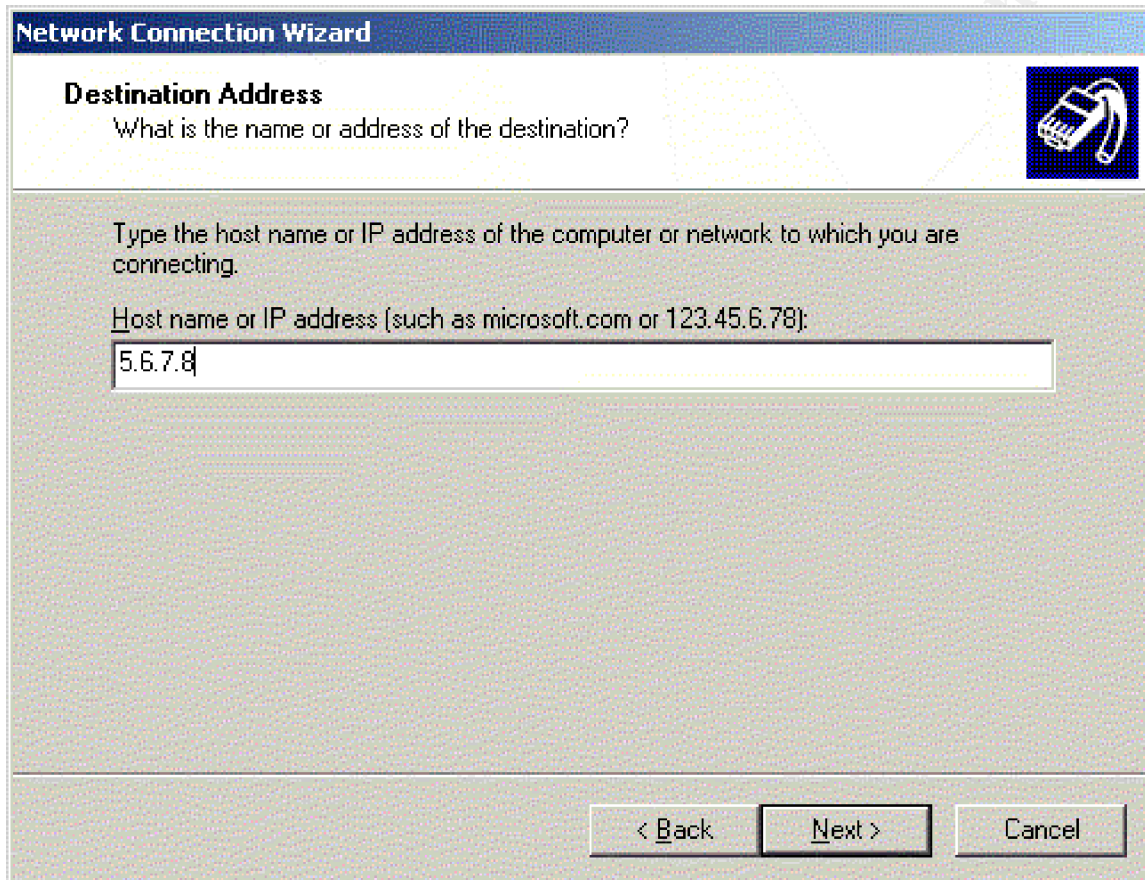


Figure 46

We are obviously setting up a VPN, so we select that option and click Next. We

are then asked for the name or IP address of the server we want to connect to; we could enter a fully qualified domain name, such as vpn.giac.com, but since we know the IP address and don't expect it to change any time soon, we'll just enter it here. This has the slight advantage of not needing to rely on DNS to connect successfully.



The screenshot shows a Windows XP-style dialog box titled "Network Connection Wizard". The main heading is "Destination Address" with a sub-question "What is the name or address of the destination?". Below this, it says "Type the host name or IP address of the computer or network to which you are connecting." There is a text input field labeled "Host name or IP address (such as microsoft.com or 123.45.6.78):" which contains the text "5.6.7.8". At the bottom right, there are three buttons: "< Back", "Next >", and "Cancel". A small icon of a network card is in the top right corner.

Figure 47

We then have the choice of making this connection available to all users, or just ourselves. As staff with desktop PC's do occasionally borrow machines from our "pool" of communal laptops, we configure it for all users.

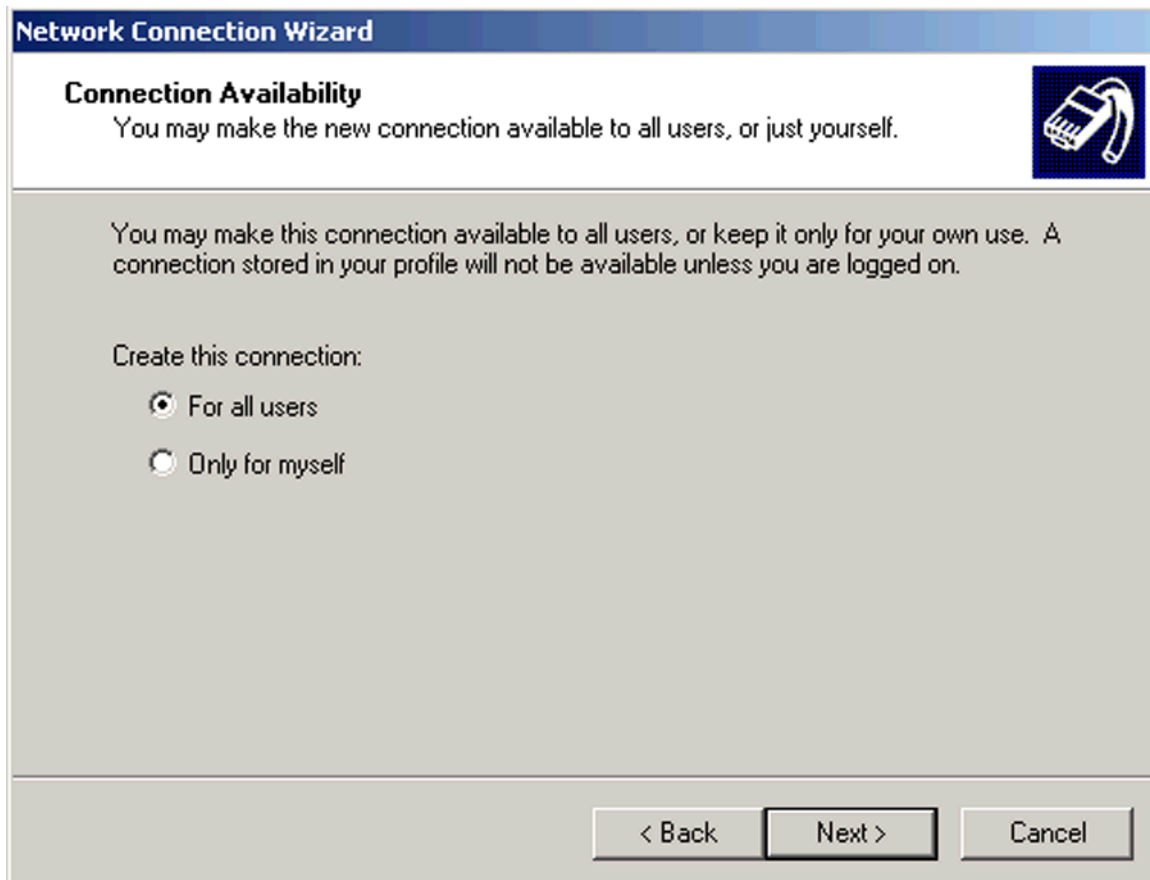


Figure 48

That's it for the wizard, aside from naming our connection. Windows defaults to Virtual Private Connection; we change it to "Outlook Synchronization via Internet."





Figure 49

Our L2TP VPN is not yet ready to go, though. We still need to confirm the properties of our resulting connection, so we open it up and click on Properties. On the resultant screen, we click on the Security tab, and click to configure Advanced settings. Again, we're only using MS-CHAP v2.

Here is an example of what we see in our server's Security logs (and, of course, on our syslog server) after a successful IPSec connection. There are many other events logged along with this, but this one shows us the juicy details about our IKE session.

```
Event Type: Success Audit
Event Source: Security
Event Category: System Event
Event ID: 541
Date: 10/9/2002
Time: 12:00:10 PM
User: BUILTIN\Administrators
Computer: L2TP
Description:
IKE security association established.
Mode:
Data Protection Mode (Quick Mode)
```

Peer Identity:  
Certificate based Identity.  
Subject test-client.giacent.com  
Issuing Certificate Authority mis@giac.com, US, DC, Washington, GIAC  
Enterprises, IT, GIAC Issuing CA  
Root Certificate Authority mis@giac.com, US, DC, Washington, GIAC  
Enterprises, IT, GIAC Root CA  
Peer IP Address: 5.6.23.147

Filter:  
Source IP Address 5.6.7.8  
Source IP Address Mask 255.255.255.255  
Destination IP Address 5.6.23.147  
Destination IP Address Mask 255.255.255.255  
Protocol 17  
Source Port 0  
Destination Port 1701

Parameters:  
ESP Algorithm Triple DES CBC  
HMAC Algorithm MD5  
AH Algorithm None  
Encapsulation Transport Mode  
InboundSpi -1763540297  
OutBoundSpi 1046241578  
Lifetime (sec) 3600  
Lifetime (kb) 250000

### **Real World implementation hints:**

Getting a VPN connection to work can be pretty difficult. My suggestion, to help ease implementation and troubleshooting, is to follow these steps:

- 1) After configuring both the server and a client, test them without any intervening routers or firewalls.
- 2) After you've got step 1 working, add the firewall machine, but with filtering turned off and routing enabled. If you can't get it to work without filtering, you'll never get it to work with it! Watch out for the routing tables on not just the firewall/router and the VPN server, but the client as well. (We found we had to edit the routing table on our VPN server to get it to work, as the routes it set up by default didn't know what to do with replies to our external connection.)
- 3) After step 2 is working, *then* you can turn on your packet filtering and see if it still works.

In summary, we're configuring our VPN server's interfaces to allow only the same traffic that our primary firewall allows. We do this with both the Symantec firewall software and the IP filters within RRAS; the only difference between the



firewall and the IP filters is that the firewall allows ESP (IP protocol 50) for IPSec.

Filters for the external interface:

1. Allow inbound UDP destined to port 1701, and replies.
2. Allow inbound UDP destined to port 500, and replies.
3. Allow IP protocol 50 (firewall only.)

That's it. Since we're using digital certificates to establish the IPSec connections, and the opportunity to enter a username and password doesn't occur until after the IPSec connection is successful, we don't have to worry about people locking out our accounts simply by banging away at the VPN.

Filters for the internal interface. (Replies to all of these are allowed as well.) Unless otherwise specified, ports are TCP.

1. Outbound 88 (Kerberos)
2. Outbound 135 (RPC portmapper)
3. Outbound UDP 137-138 (login sequence, NetBIOS)
4. Outbound 139 (login sequence, NetBIOS)
5. Outbound 389 (LDAP, DC's only)
6. Outbound 445 (NetBIOS)
7. Outbound 3268 (Global Catalogue, DC's only)
8. Outbound 4444-4446 (Exchange server only)
9. Outbound 5555-5655

## **B. Citrix NFuse v. 1.7**

Citrix NFuse, in our case, runs on Microsoft Internet Information Server 5.0 (IIS 5). This is certainly cause for alarm, as the number of security flaws, and working exploits for them, is legion. Hardening IIS and making it reasonably secure is not impossible, however. We don't delude ourselves into believing we're safe from Denial of Service attacks, but have taken all the steps we can to minimize, to the greatest extent possible, the odds of our server being compromised. As with the firewall and RRAS descriptions above, a HOW-TO for hardening IIS is beyond the scope of this document. There is a nice overview at <http://www.secadministrator.com/articles/index.cfm?ArticleID=22365&pg=1>, "Secure Web Server Installation on Win2K: Create a bastion host IIS machine." The fundamental concepts, however are as follows:

1. Harden the underlying operating system (i.e. Windows 2000 Server) by disabling any services not absolutely required, make registry edits to harden the TCP/IP stack, delete unnecessary files, apply service packs and hotfixes, etc.
2. Use NTFS permissions to lock down access to the hard drives, registry

keys, etc.

3. Place the files presented by the Web server (i.e., the Website itself) on a separate partition from the operating system. Note that Citrix NFuse does not offer a choice as to where it installs itself -- it always goes to the default folder on the C: drive. These files can, and should, be moved; however, there are scripts that must be run afterwards to update the NFuse configuration files; these are available from Citrix. See [http://hgextsrvsft01.citrix.com/cgi-bin/webcgi.exe/?Session=4892759,U=1,ST=96,N=0005,K=29837,SXI=12,Cas e=obj\(4043\)](http://hgextsrvsft01.citrix.com/cgi-bin/webcgi.exe/?Session=4892759,U=1,ST=96,N=0005,K=29837,SXI=12,Cas e=obj(4043)), "NFuse Classic 1.7 Error: Blank page at redirect.asp after moving web pages" for details.

4. Run the available Microsoft utilities, such as the IIS Lockdown tool (<http://www.microsoft.com/downloads/release.asp?ReleaseID=33961&area=search&ordinal=2>).

This is by no means a complete listing of the tasks required. If you're running an IIS Web server, expect to invest a considerable amount of time locking it down.

Also note that performing all of these tasks is by no means a guarantee you're safe or secure. However, it can go a very long way; servers that were properly locked down were not infected with Code Red or Nimda even if they didn't have the corresponding patches applied yet.<sup>20</sup>

We will be installing Citrix Secure Gateway (CSG) on this same server. Because both CSG and our NFuse Website will be using HTTPS/SSL (tcp port 443), we will have to jump through a few hoops to make this work. This process is described at [http://hgextsrvsft01.citrix.com/cgi-bin/webcgi.exe/?Session=4822373,U=1,ST=187,N=0005,K=1249,SXI=12,Cas e=obj\(13815\)](http://hgextsrvsft01.citrix.com/cgi-bin/webcgi.exe/?Session=4822373,U=1,ST=187,N=0005,K=1249,SXI=12,Cas e=obj(13815)), "Running Citrix Secure Gateway and IIS/NFuse on the same server." In a nutshell, the problem is IIS binds itself to all network interfaces by default; see Microsoft Knowledge Base article Q238131, "How to Disable Socket Pooling."<sup>21</sup>

Connecting to the NFuse site will be a very simple matter for staff; they will simply have to point their browsers to office.giac.com. Because they will be sending their usernames and passwords across the Internet, this site will use 128 bit encryption using the RSA SHA-1 algorithm. While this is no guarantee, it should provide sufficient security for GIAC Enterprises as the costs involved in cracking the cipher would be prohibitive.

As with everything else, our policy will be to allow only those services that are required. This means that staff will have access to the applications they need,

<sup>20</sup> See Mark Minasi's "Windows 2000/NT Newsletter Issue #16 August 2001" in the archives at <http://www.minasi.com/showdoc.asp?docname=nws0108.htm>, noting in particular the section titled "How I Mostly Avoided Code Red Problems." Note that this site requires free registration.

<sup>21</sup> <http://support.microsoft.com/default.aspx?scid=kb:EN-US;q238131>, "How to Disable Socket Pooling"

nothing more and nothing less.

The personal firewall rulesets for this server are pretty simple. The Internet side allows only TCP ports 80 and 443; the DMZ side needs 443 and 1494 (Citrix ICA.)

## C. SSH-2

GIAC Enterprises' IT staff will use SSH-2 for remotely accessing servers in the DMZ, the primary firewall, and the internal network. This will be done in all cross-subnet cases by first using SSH to connect to a dedicated SSH server located in the DMZ; this will be true whether the access is from the internal network to the DMZ, firewall, or Internet (for managing a staff members' home firewall, when that capability is implemented) or when they're accessing anything on the network from the Internet. In other words, the only SSH traffic permitted by the firewall in any direction will be to or from the DMZ's SSH server. Once authenticated to this "proxy" server, staff will then initiate SSH sessions to the servers being managed.

This will be done using OpenSSH version 3.5<sup>22</sup>, using Blowfish<sup>23</sup> for encryption (with 128 bit keys) because it's fast and secure enough for our needs.

Key distribution will be closely monitored. There was some brief discussion of using the same Certificate Authority servers GIAC Enterprises uses for Windows 2000, but pursuing that has been put off until there's time to deal with it; it's not a big enough hassle maintaining them separately to be worth the hassle of attempting to integrate.

We will not be including a tutorial on SSH implementation here, but we do want to reiterate that we will not even install SSH-1 capability because of the inherent security flaws. We should also point out that, in accordance with our security policy, we apply patches and updates as they become available; the recent security alert<sup>24</sup> for SSH-2 enabling privilege escalation has, for example, been addressed and patched.

The *Official Red Hat Linux 7.3 Reference Guide*<sup>25</sup> also has a very nice, detailed explanation of how to configure OpenSSH.

---

<sup>22</sup> <http://www.openssh.org/portable.html>

<sup>23</sup> <http://www.counterpane.com/blowfish.html>

<sup>24</sup> <http://www.openssh.org/txt/iss.adv>

<sup>25</sup> <http://www.redhat.com/docs/manuals/linux/RHL-7.3-Manual/ref-guide/s1-ssh-configfiles.html>

# Assignment 3: Verify the Firewall Policy

## Overview

We need to make sure our primary firewall (Firewall A) properly enforces our policy. There will be 3 primary parts to our firewall audit. Obviously, the first step is to plan the audit, since without a detailed list of tasks and methods, we'll be floundering aimlessly, never knowing when we're finished. Secondly, we have to actually conduct the audit, following our plan. Lastly, we'll evaluate our work; this is, of course, the whole point. If we find in this third part of the audit that our firewall does not perform as intended, or in any other way does not correctly enforce our policies outlined in Assignment 1, we will probably have to make adjustments to the firewall -- but it is theoretically possible that we will instead need to re-evaluate our policies, as there has not yet been an audit of the policies themselves. We will operate under the assumption, however, that any failures of our firewall to properly enforce our policies will be corrected by changes to the firewall. Also note that if we make changes to our firewall, we will have to re-start our audit from step 1.

## 1. The Plan

Our policy requires providing services as well as blocking ports, so we need to test to ensure the services we need actually work, as well as test to ensure we can't get to anything we don't need. This would ideally be done through the use of netcat,<sup>26</sup> to simulate traffic and loads. However, our upper management is skeptical of "fake" traffic, and has requested we actually use each service in question (for example, by connecting to the Citrix NFuse server and launching applications.) If we were auditing more than just our firewall, we would also utilize tcpdump to verify that our traffic is, indeed, encrypted as intended, since we place little faith in the existence of the little yellow key icon in Internet Explorer. (For example, we would log into the Citrix NFuse server and analyze the traffic.) However, our assignment here is to audit the firewall, not the entire policy implementation; as such, we won't be performing such in-depth traffic analysis at this point.

Naturally, any usernames & passwords used during this testing will be temporary accounts set up for this purpose and deleted after the audit. We will be further testing our firewall's implementation of our (intended) rulesets primarily through the use of nmap<sup>27</sup> scans. We use nmap because, as stated by McClure *et al*:

"When nmap scans a host, it doesn't just tell you which ports are open or closed,

<sup>26</sup> netcat can be found at <http://www.atstake.com/research/tools/nc110.tgz>

<sup>27</sup> nmap can be found at <http://www.insecure.org/nmap/>

it tells you which ports are being blocked. The amount (or lack) of information received from a port scan can tell a lot about the configuration of the firewall.”<sup>28</sup>

Also, nmap is the tool most likely to be used against us in the real world, so it's important to know how our firewall reacts to it.

Our audit will therefore take the following course:

A. First, we need to do a simple “sanity” check. We will re-read our policy, then proofread our firewall rulesets on paper. We will not only be looking for typo's, which could have disastrous results in our firewall's functionality, but we will also be double-checking to make sure the rules we've written look like they should be implementing our policies. We will simply be asking ourselves, “Does what we wrote make sense?”

This can be done during normal office hours, because it will not disrupt the office network in any way. It should be completed within a single day.

B. Because our firewall boots off of a CD, and runs on an older PC model we have multiple copies of, we will begin the actual testing of rulesets by setting up a test firewall. Because it will be on identical hardware, running the same, identical configuration, we should get accurate results. This will enable us to perform scans and tests against our firewall without disrupting the office staff in any way. This will act as the second stage of our “sanity” check, ensuring the ports we think we've opened are, in fact, open, and all the ports we did not intend to open are in fact closed. Because the test machine is not connected to a “live” network, we will be able to scan our firewall on each of its 5 network interfaces, including the 2 used by the L2TP VPN server. We will, however, connect machines to each interface to respond to any scans that get through. We need these machines on each interface because nmap treats scans that receive no response as either filtered or open depending on the type of connection. For example, a TCP SYN scan that receives no response is considered a filtered port; a UDP scan that receives no response is considered open. If we scan our firewall without anything connected to it, the firewall wouldn't have anyplace to send a SYN packet to; since our firewall's ICMP filters prevent host unreachables, our scans would never receive a reply and nmap would report all the ports as filtered.

This can be done during normal office hours, because it will not disrupt the office network in any way. This also enables us to use fairly aggressive scans without concern for disrupting our legitimate traffic, though we will also want to perform some more “stealthy” scans for comparison; nmap sometimes seems to have better success that way. Because of the desire to use these slower, less obtrusive scans, this step will take several days to complete. However, most of that time will not require any staff time, because it will simply be a matter of waiting for the scans to run.

---

<sup>28</sup> *Hacking Exposed, Third Edition*, Stuart McClure *et al*, p. 485

Assuming we haven't found anything wrong [a risky assumption, given the complexity of our rulesets!], we will then move on to step C. If we do find any problems, we will address them and restart from step A above, repeating steps A and B until we've eliminated all the errors.

C. Testing the functionality of services is the next step. This will have to be done on the "live" network, which means we need to be more conscious of when we perform our testing. While simply connecting to a service should not disrupt office work in any way -- and, in fact, we need to ensure accessing the services during normal business hours doesn't present our network with any performance problems -- if we were performing a full audit we would want to perform some of our tests off-hours to improve the accuracy of our analysis. Using the example cited above, if we tried to perform tcpdump scans of our Citrix NFuse logon while staff were using it, there would be much more effort required to sift through the scans looking for our relevant traffic. Again, we're only testing our firewall now though, so those off-hour tests are not necessary.

The list of services we need to test functionality for is as follows:

1. Citrix NFuse with Citrix Secure Gateway
2. L2TP over IPSec, with Outlook synchronization
3. SSH-2
  - a. from outside the network to the DMZ
  - b. from inside the network to the DMZ
  - c. from the DMZ to inside and outside the network.
4. The GIAC.COM Website
5. Accessing the World Wide Web from the internal office.
6. Passive FTP from the internal office
7. Sending and receiving e-mail
8. Our Syslog servers in both the DMZ and the internal networks
9. Out Snort IDS systems. (While this is not directly related to the firewall, we thought it best to check them while the types of normal traffic patterns we're dealing with are fresh in our minds.)

We will ensure our services work manually. While this is a laborious, low-tech method of testing, it has been requested by some in upper management, who don't understand netcat and, therefore, don't trust it.

Because of the need to do things manually, some of which must be done from outside of the network, this will be a fairly lengthy endeavor, taking several days to complete. The IT staff recognizes the need to accurately log their work; they will have to use screenshots as much as possible to verify things for upper management. (These screenshots are not, however, included in this paper.)

D. The next step will be to review the work done to check for completeness. This will be, in effect, a mini-audit of our audit. Because it will simply be a

matter of comparing work with checklists, this should only take a couple of hours at most to complete.

E. Once we are confident the work was thorough, we will then evaluate the audit. This will involve comparing expected results with actual results. Discrepancies or problems will be identified.

F. Lastly, we will make recommendations for any changes that would improve security.

### **Audit costs and risks:**

The total costs for the audit are difficult to judge. The use of an independent company to perform the audit was obviously not approved, so the work will be done by GIAC Enterprises' own IT staff. This fact, combined with the fact that the tools used are free, means the only costs involved should be the time involved. Assuming all goes well, billable labor time is estimated to be less than one week, though because of the time required for some of the scans the process may actually take as much as three weeks.

The risks involved with this audit are primarily that we may temporarily disrupt GIAC Enterprises' business; some of our nmap scans will be sending malformed packets, which can cause some TCP/IP stacks to crash. We don't expect that reaction from either Red Hat or Windows 2000, so this risk is deemed low and acceptable. The risks involved with sending information over the Internet will be minimized by using temporary accounts created solely for the purpose of testing. A separate account will be created and used for each test; these accounts will then be deleted immediately upon test completion.

In addition, some ISP's have guidelines that prohibit the use of scanning tools from within their networks. Because our scans will be done on a test server, this should not be a problem; care will be taken, however, to ensure any scans over the Internet that are determined necessary are done only from suitable locations.

## **2. Conducting the Audit**

A. As stated in the Plan, we will begin with a "sanity" check. This resulted in a couple of corrections to our firewall script<sup>29</sup>:

1. We had forgotten the Citrix NFuse server in the DMZ needs access to the Citrix MetaFrame servers in the internal network in order to authenticate user logons. This is done via the Citrix SSL Relay service, so we had to open up tcp port 443.

---

<sup>29</sup> These changes are already been reflected in the corrected script presented in Assignment 2 above.

2. We had configured DNS traffic from the L2TP VPN server to go to the DMZ DNS server, protecting our internal server. However, because the DNS queries coming from the VPN will be for servers located in the internal network, about which our DMZ DNS server knows nothing, we had to change the rules. The firewall now allows DNS access (UDP 53) for the L2TP VPN server to the internal DNS server.

3. Several “typos,” such as single dashes instead of double.

**B.** Our next step is to scan the various interfaces, using nmap. This first requires adding enough NIC’s to our test machine, then popping our bootable CD into the PC and starting it up. We do our scans from a Red Hat Linux 7.3 workstation; the only machines on this test network are the firewall, one machine on each interface running netcat in listening mode to mimic the services provided on that interface, and the scanning workstation. We connect our scanning workstation directly to the interface being scanned via cross-over cable. We will be performing multiple scans here, in the hopes of being as thorough as possible.

Each “listening” workstation had IP aliases set up in its ifcfg-eth0 file enabling it to listen on all the required ip addresses. We also set up netcat in listening mode, so there is something listening in the event our scans get through the firewall. Note that we want to listen to all ports on all assigned ip addresses, so our scripts are pretty simple. We do have to limit the number of listening ports, however, as our machines run out of resources if we open up too many. We’ll open up all the ports we need, plus enough extras to ensure any openings get noticed. (Remember, we aren’t programmers, so no snide comments about the script are necessary; we have no doubts there’s a more efficient way to do this, but this is easy to read and it works.)

Note that the “listening” script for any given interface will not be used when that is the interface being scanned.

The first “listening” script is for the external interface.

# External interface listening script

# First, set up our IP alias addresses

/sbin/ifconfig eth0:0 5.6.7.1/24 5.6.7.9 # Router

/sbin/ifconfig eth0:1 5.6.7.10/24 5.6.7.9 # WWW & SMTP servers’ address

/sbin/ifconfig eth0:2 5.6.7.11/24 5.6.7.9 # Citrix NFuse server

/sbin/ifconfig eth0:3 5.6.7.12/24 5.6.7.9 # CSG, SSH, Syslog servers

/sbin/ifconfig eth0:4 5.6.7.7/24 5.6.7.9 # Source NAT address

# Next, start listening

PORT=1

while [ \$PORT -lt 1025 ]; do

nc -l -p \$PORT &

---



```

    let PORT=PORT+1
done

nc -l -p 1494 &
nc -l -p 1701 &
nc -l -p 1723 &

PORT=4444

while [ $PORT -lt 4447 ]; do
    nc -l -p $PORT &
    let PORT=PORT+1
done

PORT=5555

while [ $PORT -lt 5700 ]; do
    nc -l -p $PORT &
    let PORT=PORT+1
done

```

---

Here's the script for listening on the Internal interface.

```

# Internal interface listening script

# First, set up our IP addresses
/sbin/ifconfig eth0:0 10.0.0.15/24 10.0.0.1 # SMTP server
/sbin/ifconfig eth0:1 10.0.0.34/24 10.0.0.1 # IT staff PC
/sbin/ifconfig eth0:2 10.0.0.29/24 10.0.0.1 # Citrix Metaframe server #1
/sbin/ifconfig eth0:3 10.0.0.30/24 10.0.0.1 # Citrix Metaframe server #2
/sbin/ifconfig eth0:4 10.0.0.131/24 10.0.0.1 # Staff PC
/sbin/ifconfig eth0:5 10.0.0.8/24 10.0.0.1 # Syslog server
/sbin/ifconfig eth0:6 10.0.0.21/24 10.0.0.1 # DC1 server
/sbin/ifconfig eth0:7 10.0.0.22/24 10.0.0.1 # DC2 server
/sbin/ifconfig eth0:8 10.0.0.23/24 10.0.0.1 # DC3 server
/sbin/ifconfig eth0:9 10.0.0.17/24 10.0.0.1 # SUS server

# Next, start listening

PORT=1

while [ $PORT -lt 1025 ]; do
    nc -l -p $PORT &
    let PORT=PORT+1
done

nc -l -p 1494 &
nc -l -p 1701 &
nc -l -p 1723 &

PORT=4444

while [ $PORT -lt 4447 ]; do
    nc -l -p $PORT &
    let PORT=PORT+1

```

---

```
done
```

```
PORT=5555
```

```
while [ $PORT -lt 5700 ]; do  
    nc -l -p $PORT &  
    let PORT=PORT+1  
done
```

---

We also need a script for our DMZ mimic.

```
# DMZ interface listening script
```

```
# First, set up our IP addresses  
/sbin/ifconfig eth0:0 10.0.1.2/24 10.0.1.1 # WWW server  
/sbin/ifconfig eth0:1 10.0.1.9/24 10.0.1.1 # NFuse server  
/sbin/ifconfig eth0:2 10.0.1.6/24 10.0.1.1 # CSG server  
/sbin/ifconfig eth0:3 10.0.1.11/24 10.0.1.1 # SSH server  
/sbin/ifconfig eth0:4 10.0.1.15/24 10.0.1.1 # SMTP relay  
/sbin/ifconfig eth0:5 10.0.1.8/24 10.0.1.1 # Syslog, NTP server  
/sbin/ifconfig eth0:6 10.0.1.7/24 10.0.1.1 # Squids proxy  
/sbin/ifconfig eth0:7 10.0.1.3/24 10.0.1.1 # DNS server  
/sbin/ifconfig eth0:8 10.0.1.28/24 10.0.1.1 # SQL server  
/sbin/ifconfig eth0:9 10.0.1.17/24 10.0.1.1 # SUS, NAV server
```

```
# Next, start listening
```

```
PORT=1
```

```
while [ $PORT -lt 1025 ]; do  
    nc -l -p $PORT &  
    let PORT=PORT+1  
done
```

```
nc -l -p 1494 &  
nc -l -p 1701 &  
nc -l -p 1723 &
```

```
PORT=4444
```

```
while [ $PORT -lt 4447 ]; do  
    nc -l -p $PORT &  
    let PORT=PORT+1  
done
```

```
PORT=5555
```

```
while [ $PORT -lt 5700 ]; do  
    nc -l -p $PORT &  
    let PORT=PORT+1  
done
```

---

We don't need to add the IP aliases for the L2TP VPN interfaces, since there's only one IP address connected to each interface; all we need is the listening script. For both of these interfaces, this is identical to the "start listening" section of the scripts above, so we won't repeat it here.

In order to do our scans as logically as possible, we wrote a simple script for scanning each interface we intend to scan. The bulk of the scans will be the same from each interface, though we'll have to customize them a little bit for the scans in which we spoof the address we're scanning from.

One limitation to what we're doing here is that nmap does not scan ALL ports by default, only those known to have services associated with them. This is because scans would otherwise take much, much longer to complete. We won't waste our time with such scanning unless we see something that raises suspicions.

We start with the external interface. Since we're doing this sitting next to the firewall, our job is a little easier; we just set our network interface to the ip address we want (in this case, 5.6.7.1). We'll add aliases in some of our other scripts, but for the External interface that's all we need.

### **External Interface Scanning Script:**

```
# External Interface Firewall Scanning Script

# Part 1: scanning the subnet

# First, do a standard TCP scan. We use the following switches:
#
# -n This tells nmap not to resolve IP addresses to names;
#     it's a little bit faster, plus we don't need names
#
# -r This tells nmap NOT to randomize port scans; this makes
#     our log review a bit easier
#
# -vv This tells nmap to give us detailed, verbose logs
#
# -sT We're doing a "TCP Connect" scan; very noisy & obvious
#
# -O This tells nmap to attempt OS identification
#
# -P0 We don't need to do Pings; in fact, nmap will tell us the target
#     host is unavailable if we don't include this.

# -oN Log in human-readable format in the filename that follows
```

```

#
# 5.6.7.2,7-12 The addresses we're scanning

nmap -n -r -vv -sT -O -P0 -oN eth0-sT.log 5.6.7.2,7-12 &&

# Next, we make the following changes:
#
# -sS We're doing a SYN scan now, not a full connect

nmap -n -r -vv -sS -P0 -oN eth0-sS.log 5.6.7.2,7-12 &&

# Now we try an ACK scan, using the -sA switch. This is a good way to
# see if a firewall is stateful or not.

nmap -n -r -vv -sA -P0 -oN eth0-sA.log 5.6.7.2,7-12 &&

# Perform a FIN scan, using -sF; this is a bit stealthier than -sS

nmap -n -r -vv -sF -P0 -oN eth0-sF.log 5.6.7.2,7-12 &&

# Try a "Christmas Tree" scan; this sends packets with FIN, PSH, and
# URG. Note that this probably won't show different port results than
# the FIN scan, but we want to see how our firewall reacts to it; we're
# expecting to see it on the firewall's logs.

nmap -n -r -vv -sX -P0 -oN eth0-sX.log 5.6.7.2,7-12 &&

# Null scan, using -sN. As with -sX, we're mostly just testing the
# firewall's reaction

nmap -n -r -vv -sN -P0 -oN eth0-sN.log 5.6.7.2,7-12 &&

# IP Protocol scan. Any non-response is viewed as an open port by nmap,
# so this should try to tell us all protocols are allowed...

nmap -n -r -vv -sO -P0 -oN eth0-sO.log 5.6.7.2,7-12 &&

# Ping scan. This takes the -sP option; sends ACK's as well.
# Note that the -P0 option wouldn't make sense here...

nmap -n -r -vv -sP -oN eth0-sP.log 5.6.7.2,7-12 &&

# UDP scan. Unreliable; like -sO, all non-rejects are assumed open.

nmap -n -r -vv -sU -P0 -oN eth0-sU.log 5.6.7.2,7-12 &&

# Now that we've completed our "generic" scans, we try some more
# directed ones.
# The -g option lets us specify a source port for our scans to come from.
# We'll check HTTP, HTTPS, SMTP, SSH, PPTP, ICA, L2TP, DNS, syslog, IKE,
# and NTP.
# For comparison, we'll also try RPC (TCP 135) and ICA (TCP 1494), which

```

# we allow in some places but not on this interface. Lastly, note that  
# these are pretending to be replies, so we want -sA or -sU.

```
nmap -n -r -vv -sA -P0 -g 80 -oN eth0-sport80.log 5.6.7.2,7-12 &&  
nmap -n -r -vv -sA -P0 -g 443 -oN eth0-sport443.log 5.6.7.2,7-12 &&  
nmap -n -r -vv -sA -P0 -g 21 -oN eth0-sport21.log 5.6.7.2,7-12 &&  
nmap -n -r -vv -sA -P0 -g 25 -oN eth0-sport25.log 5.6.7.2,7-12 &&  
nmap -n -r -vv -sA -P0 -g 22 -oN eth0-sport22.log 5.6.7.2,7-12 &&  
nmap -n -r -vv -sA -P0 -g 1723 -oN eth0-sport1723.log 5.6.7.2,7-12 &&  
nmap -n -r -vv -sU -P0 -g 500 -oN eth0-sport500.log 5.6.7.2,7-12 &&  
nmap -n -r -vv -sU -P0 -g 1701 -oN eth0-sport1701.log 5.6.7.2,7-12 &&  
nmap -n -r -vv -sU -P0 -g 53 -oN eth0-sport53.log 5.6.7.2,7-12 &&  
nmap -n -r -vv -sU -P0 -g 514 -oN eth0-sport514.log 5.6.7.2,7-12 &&  
nmap -n -r -vv -sU -P0 -g 123 -oN eth0-sport123.log 5.6.7.2,7-12 &&  
nmap -n -r -vv -sA -P0 -g 1494 -oN eth0-sport1494.log 5.6.7.2,7-12 &&  
nmap -n -r -vv -sA -P0 -g 135 -oN eth0-sport135.log 5.6.7.2,7-12 &&
```

# The last scan we want to do of this subnet will be a slower, more  
# stealthy scan, to see if patience gives us anything extra.  
# We do this by adding the -T Sneaky option, which waits 15 seconds  
# between scans.  
# This will take quite a while to run!

```
nmap -n -r -vv -sS -P0 -T Sneaky -oN eth0-sS-Sneaky.log 5.6.7.2,7-12 &
```

# Part 2 - other subnets

#

# Since we know our firewall is acting as a router, we will try to scan  
# the subnets on the other side of it. We expect these to fail.

#

# We'll just do a couple here; if any of these give us anything, then  
# we'll have to do perform a much more thorough examination to find out  
# what's wrong.

# First, set up our routing table

```
route add -net 10.0.1.0/24 gw 5.6.7.9 # DMZ  
route add -net 10.0.0.0/24 gw 5.6.7.9 # Internal network  
route add -net 10.0.2.0/24 gw 5.6.7.9 # L2TP VPN's internal interface
```

# Next, try to reach a machine on each subnet

```
nmap -n -r -vv -sS -P0 -oN eth0-DMZ-sS.log 10.0.1.2 && # Web server  
nmap -n -r -vv -sS -P0 -oN eth0-INT-sS.log 10.0.0.21 && # DC server  
nmap -n -r -vv -sS -P0 -oN eth0-VPN-sS.log 10.0.2.12 && # L2TP VPN
```

# That's it for this interface (we hope...) Note that we'll do  
# some more scanning on the live firewall as well, just to be sure.

## External Interface Scan Results and Analysis:

Mostly good results. Our logs successfully notified us of the invalid TCP flag scans, as well.

We see here we were unable to reach the firewall's NIC attached to the L2TP VPN's external interface (5.6.7.2), the Source NAT'ed address (5.6.7.7), or the firewall's external interface (5.6.7.9).

```
# nmap (V. 2.54BETA31) scan initiated Thu Oct 10 10:41:25 2002 as: nmap -n -r -vv -sT -O -P0 -oN eth0-sS.log 5.6.7.2,7-12
```

Warning: OS detection will be MUCH less reliable because we did not find at least 1 open and 1 closed TCP port

All 1554 scanned ports on (5.6.7.2) are: filtered

Too many fingerprints match this host for me to give an accurate OS guess

TCP/IP fingerprint:

SInfo(V=2.54BETA31%P=i386-redhat-linux-gnu%D=10/10%Time=3DA595CD%O=-1%C=-1)

T5(Resp=N)

T6(Resp=N)

T7(Resp=N)

PU(Resp=N)

Warning: OS detection will be MUCH less reliable because we did not find at least 1 open and 1 closed TCP port

All 1554 scanned ports on (5.6.7.7) are: filtered

Too many fingerprints match this host for me to give an accurate OS guess

TCP/IP fingerprint:

SInfo(V=2.54BETA31%P=i386-redhat-linux-gnu%D=10/10%Time=3DA59A04%O=-1%C=-1)

T5(Resp=N)

T6(Resp=N)

T7(Resp=N)

PU(Resp=N)

Warning: OS detection will be MUCH less reliable because we did not find at least 1 open and 1 closed TCP port

All 1554 scanned ports on (5.6.7.9) are: filtered

Too many fingerprints match this host for me to give an accurate OS guess

TCP/IP fingerprint:

SInfo(V=2.54BETA31%P=i386-redhat-linux-gnu%D=10/10%Time=3DA5A150%O=-1%C=-1)

T5(Resp=N)

T6(Resp=N)

T7(Resp=N)

PU(Resp=N)

Even without any communication, nmap accurately guessed we were running Red Hat Linux.

Our scans of the other publicly accessible ports provided the expected results. Our scan of the 5.6.7.12 address, for example shows this:

Port	State	Service
22/tcp	open	ssh
25/tcp	open	smtp

443/tcp open https

Our attempts to scan the other, private subnets from here failed as expected. The logged output from our scan of the domain controller shows this:

```
# nmap (V. 2.54BETA31) scan initiated Thu Oct 10 10:13:21 2002 as: nmap -e eth0 -n -r -vv -sS -P0 -oN eth0-INT-sS.log 10.0.0.21
All 1554 scanned ports on (10.0.0.21) are: filtered
```

```
# Nmap run completed at Thu Oct 10 10:20:23 2002 -- 1 IP address (1 host up) scanned in 1680 seconds
```

Next, we scanned the internal interface. We simply scan all three subnets (DMZ, VPN, and public address space).

### Script for scanning the internal interface:

```
# Internal Interface Firewall Scanning Script
```

```
# Before we start scanning, we need to set up our computer to be able to
# respond to the IP addresses to use
```

```
/sbin/ifconfig eth0:0 10.0.0.15/24 10.0.0.1 # Our SMTP server
/sbin/ifconfig eth0:1 10.0.0.17/24 10.0.0.1 # Our SUS server
/sbin/ifconfig eth0:2 10.0.0.21/24 10.0.0.1 # A DC server
/sbin/ifconfig eth0:3 10.0.0.29/24 10.0.0.1 # A Citrix server
/sbin/ifconfig eth0:4 10.0.0.34/24 10.0.0.1 # An IT staff's PC
/sbin/ifconfig eth0:5 10.0.0.131/24 10.0.0.1 # A staff PC
```

```
# First, do a standard TCP scan. We use the following switches:
```

```
#
```

```
# -n This tells nmap not to resolve IP addresses to names;
```

```
# it's a little bit faster, plus we don't need names
```

```
#
```

```
# -r This tells nmap NOT to randomize port scans; this makes
```

```
# our log review a bit easier
```

```
#
```

```
# -vv This tells nmap to give us detailed, verbose logs
```

```
#
```

```
# -sT We're doing a "TCP Connect" scan; very noisy & obvious
```

```
#
```

```
# -O This tells nmap to attempt OS identification
```

```
#
```

```
# -P0 We don't need to do Pings; in fact, nmap will tell us the target
```

```
# host is unavailable and quit if we don't include this.
```

```
# -oN Log in human-readable format in the filename that follows
```

```
#
```

```
# 10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 - The addresses we're scanning
```

```
nmap -n -r -vv -sT -O -P0 -oN eth1-sT.log \
```

```

10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&

# Next, we make the following changes:
#
# -sS We're doing a SYN scan now, not a full connect

nmap -n -r -vv -sS -P0 -oN eth1-sS.log \
    10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&

# Now we try an ACK scan, using the -sA switch. This is a good way to
# see if a firewall is stateful or not.

nmap -n -r -vv -sA -P0 -oN eth1-sA.log \
    10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&

# Perform a FIN scan, using -sF; this is a bit stealthier than -sS

nmap -n -r -vv -sF -P0 -oN eth1-sF.log \
    10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&

# Try a "Christmas Tree" scan; this sends packets with FIN, PSH, and
# URG. Note that this probably won't show different port results than
# the FIN scan, but we want to see how our firewall reacts to it; we're
# expecting to see it on the firewall's logs.

nmap -n -r -vv -sX -P0 -oN eth1-sX.log \
    10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&

# Null scan, using -sN. As with -sX, we're mostly just testing the
# firewall's reaction

nmap -n -r -vv -sN -P0 -oN eth1-sN.log \
    10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&

# IP Protocol scan. Any non-response is viewed as an open port by nmap,
# so this should try to tell us all protocols are allowed...

nmap -n -r -vv -sO -P0 -oN eth1-sO.log \
    10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&

# Ping scan. This takes the -sP option; sends ACK's as well.
# Note that the -P0 option wouldn't make sense here...

nmap -n -r -vv -sP -oN eth1-sP.log \
    10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&

# UDP scan. Unreliable; like -sO, all non-rejects are assumed open.

nmap -n -r -vv -sU -P0 -oN eth1-sU.log \
    10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&

# Now that we've completed our "generic" scans, we try some more

```



# directed ones.

# The -g option lets us specify a source port for our scans to come from.

# We'll check HTTP, HTTPS, SMTP, SSH, PPTP, ICA, L2TP, DNS, syslog, IKE,

# and NTP. We will use the -S option here to specify the source address.

# Lastly, note that these are pretending to be replies, so we want -sA

# or -sU.

```
nmap -n -r -vv -sA -P0 -S 10.0.0.131 -g 80 -oN eth1-sport80.log \
10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -S 10.0.0.131 -g 443 -oN eth1-sport443.log \
10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -S 10.0.0.131 -g 21 -oN eth1-sport21.log \
10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -S 10.0.0.34 -g 21 -oN eth1-IT-sport21.log \
10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -S 10.0.0.15 -g 25 -oN eth1-sport25.log \
10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -S 10.0.0.34 -g 22 -oN eth1-sport22.log \
10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -S 10.0.0.34 -g 1723 -oN eth1-sport1723.log \
10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sU -P0 -g 500 -oN eth1-sport500.log \
10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sU -P0 -g 1701 -oN eth1-sport1701.log \
10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sU -P0 -g 53 -oN eth1-sport53.log \
10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sU -P0 -g 514 -oN eth1-sport514.log \
10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sU -P0 -g 123 -oN eth1-sport123.log \
10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -S 10.0.0.29 -g 1494 -oN eth1-sport1494.log \
10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -S 10.0.0.34 -g 135 -oN eth1-sport135.log \
10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
```

# We also need to check the upper ports we've opened. We expect them to

# be closed, since only ESTABLISHED,RELATED traffic should be getting

# through.

```
nmap -n -r -vv -sS -P0 -p 4000-7000 -oN eth1-sS-dports4k-7k.log \
10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -p 4000-7000 -oN eth1-sA-dports4k-7k.log \
10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
```

# The last scan we want to do of this subnet will be a slower, more

# stealthy scan, to see if patience gives us anything extra.

# We do this by adding the -T Sneaky option, which waits 15 seconds

# between scans.

# This will take quite a while to run!

```
nmap -n -r -vv -sS -P0 -T Sneaky -oN eth1-sS-Sneaky.log \
```

```
10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
```

```
# That's it for the eth1 interface (we hope...)
```

## Internal Interface Scan Results and Analysis:

The results of these scans were all as expected; we'll spare you the repetitive output. See Part 3-E below.

We then scan the service network's interface (eth2). As with the Internal interface above, we simply set our script to scan each of the other 3 subnets.

## Script for scanning the DMZ:

```
# DMZ Interface Firewall Scanning Script
```

```
# Before we start scanning, we need to set up our computer to be able to  
# respond to the IP addresses to use
```

```
/sbin/ifconfig eth0:0 10.0.1.2/24 10.0.1.1 # Our WWW server  
/sbin/ifconfig eth0:1 10.0.1.6/24 10.0.1.1 # Our CSG server  
/sbin/ifconfig eth0:2 10.0.1.11/24 10.0.1.1 # The SSH server  
/sbin/ifconfig eth0:3 10.0.1.7/24 10.0.1.1 # The Squid proxy  
/sbin/ifconfig eth0:4 10.0.1.17/24 10.0.1.1 # SUS, NAV server  
/sbin/ifconfig eth0:5 10.0.1.15/24 10.0.1.1 # SMTP relay server
```

```
# First, do a standard TCP scan. We use the following switches:
```

```
#  
# -n This tells nmap not to resolve IP addresses to names;  
#   it's a little bit faster, plus we don't need names  
#  
# -r This tells nmap NOT to randomize port scans; this makes  
#   our log review a bit easier  
#  
# -vv This tells nmap to give us detailed, verbose logs  
#  
# -sT We're doing a "TCP Connect" scan; very noisy & obvious  
#  
# -O This tells nmap to attempt OS identification  
#  
# -P0 We don't need to do Pings; in fact, nmap will tell us the target  
#   host is unavailable and quit if we don't include this.
```

```
# -oN Log in human-readable format in the filename that follows
```

```
#  
# 10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 - The addresses we're scanning
```

```
nmap -n -r -vv -sT -O -P0 -oN eth2-sT.log \  
10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&
```

```
# Next, we make the following changes:
```

```

#
# -sS We're doing a SYN scan now, not a full connect

nmap -n -r -vv -sS -P0 -oN eth2-sS.log \
    10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&

# Now we try an ACK scan, using the -sA switch. This is a good way to
# see if a firewall is stateful or not.

nmap -n -r -vv -sA -P0 -oN eth2-sA.log \
    10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&

# Perform a FIN scan, using -sF; this is a bit stealthier than -sS

nmap -n -r -vv -sF -P0 -oN eth2-sF.log \
    10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&

# Try a "Christmas Tree" scan; this sends packets with FIN, PSH, and
# URG. Note that this probably won't show different port results than
# the FIN scan, but we want to see how our firewall reacts to it; we're
# expecting to see it on the firewall's logs.

nmap -n -r -vv -sX -P0 -oN eth2-sX.log \
    10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&

# Null scan, using -sN. As with -sX, we're mostly just testing the
# firewall's reaction

nmap -n -r -vv -sN -P0 -oN eth2-sN.log \
    10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&

# IP Protocol scan. Any non-response is viewed as an open port by nmap,
# so this should try to tell us all protocols are allowed...

nmap -n -r -vv -sO -P0 -oN eth2-sO.log \
    10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&

# Ping scan. This takes the -sP option; sends ACK's as well.
# Note that the -P0 option wouldn't make sense here...

nmap -n -r -vv -sP -oN eth2-sP.log \
    10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&

# UDP scan. Unreliable; like -sO, all non-rejects are assumed open.

nmap -n -r -vv -sU -P0 -oN eth2-sU.log \
    10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&

# Now that we've completed our "generic" scans, we try some more
# directed ones.
# The -g option lets us specify a source port for our scans to come from.
# We'll check HTTP, HTTPS, SMTP, SSH, PPTP, ICA, L2TP, DNS, syslog, IKE,

```

# and NTP. We'll use the -S option to specify source.  
# Lastly, note that these are pretending to be replies, so we want -sA  
# or -sU.

```
nmap -n -r -vv -sA -P0 -S 10.0.1.7 -g 80 -oN eth2-sport80.log \
10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -S 10.0.1.7 -g 443 -oN eth2-sport443.log \
10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -S 10.0.1.7 -g 21 -oN eth2-sqd-sport21.log \
10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -S 10.0.1.17 -g 21 -oN eth2-nav-sport21.log \
10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -S 10.0.1.15 -g 25 -oN eth2-sport25.log \
10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -S 10.0.1.11 -g 22 -oN eth2-sport22.log \
10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -S 10.0.1.11 -g 1723 -oN eth2-sport1723.log \
10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sU -P0 -g 500 -oN eth2-sport500.log \
10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sU -P0 -g 1701 -oN eth2-sport1701.log \
10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sU -P0 -g 53 -oN eth2-sport53.log \
10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sU -P0 -g 514 -oN eth2-sport514.log \
10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sU -P0 -g 123 -oN eth2-sport123.log \
10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -S 10.0.1.6 -g 1494 -oN eth2-sport1494.log \
10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -g 135 -oN eth2-sport135.log \
10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&
```

# The last scan we want to do of this subnet will be a slower, more  
# stealthy scan, to see if patience gives us anything extra.  
# We do this by adding the -T Sneaky option, which waits 15 seconds  
# between scans.  
# This will take quite a while to run!

```
nmap -n -r -vv -sS -P0 -T Sneaky -oN eth2-sS-Sneaky.log \
10.0.0.0/24,10.0.2.0/26,5.6.7.0/28 &&
```

# That's it for the DMZ interface (we hope...)

## DMZ Interface Scan Results and Analysis:

Like the internal interface above, we'll simply state at this point that all went well. The summary of the per-address results are shown below in Part 3-E, "Evaluating the Audit."

## Internal L2TP interface scanning script:

Here, too, we will scan the other three subnets. We'll also scan a few extra ports here, since we know there are more opened up here.

# Internal L2TP Interface Firewall Scanning Script

# First, do a standard TCP scan. We use the following switches:

#

# -n This tells nmap not to resolve IP addresses to names;

# it's a little bit faster, plus we don't need names

#

# -r This tells nmap NOT to randomize port scans; this makes

# our log review a bit easier

#

# -vv This tells nmap to give us detailed, verbose logs

#

# -sT We're doing a "TCP Connect" scan; very noisy & obvious

#

# -O This tells nmap to attempt OS identification

#

# -P0 We don't need to do Pings; in fact, nmap will tell us the target

# host is unavailable and quit if we don't include this.

# -oN Log in human-readable format in the filename that follows

#

# 10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 - The addresses we're scanning

```
nmap -n -r -vv -sT -O -P0 -oN eth3-sT.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
```

# Next, we make the following changes:

#

# -sS We're doing a SYN scan now, not a full connect

```
nmap -n -r -vv -sS -P0 -oN eth3-sS.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
```

# Now we try an ACK scan, using the -sA switch. This is a good way to

# see if a firewall is stateful or not.

```
nmap -n -r -vv -sA -P0 -oN eth3-sA.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
```

# Perform a FIN scan, using -sF; this is a bit stealthier than -sS

```
nmap -n -r -vv -sF -P0 -oN eth3-sF.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
```

# Try a "Christmas Tree" scan; this sends packets with FIN, PSH, and

# URG. Note that this probably won't show different port results than

# the FIN scan, but we want to see how our firewall reacts to it; we're

# expecting to see it on the firewall's logs.

```
nmap -n -r -vv -sX -P0 -oN eth3-sX.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
```

# Null scan, using -sN. As with -sX, we're mostly just testing the  
# firewall's reaction

```
nmap -n -r -vv -sN -P0 -oN eth3-sN.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
```

# IP Protocol scan. Any non-response is viewed as an open port by nmap,  
# so this should try to tell us all protocols are allowed...

```
nmap -n -r -vv -sO -P0 -oN eth3-sO.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
```

# Ping scan. This takes the -sP option; sends ACK's as well.  
# Note that the -P0 option wouldn't make sense here...

```
nmap -n -r -vv -sP -oN eth3-sP.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
```

# UDP scan. Unreliable; like -sO, all non-rejects are assumed open.

```
nmap -n -r -vv -sU -P0 -oN eth3-sU.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
```

# Now that we've completed our "generic" scans, we try some more  
# directed ones.

# The -g option lets us specify a source port for our scans to come from.

# We'll check HTTP, HTTPS, SMTP, SSH, PPTP, ICA, L2TP, DNS, syslog, IKE,  
# and NTP.

# Lastly, note that these are pretending to be replies, so we want -sA  
# or -sU.

```
nmap -n -r -vv -sA -P0 -g 80 -oN eth3-sport80.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -g 443 -oN eth3-sport443.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -g 21 -oN eth3-sport21.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -g 25 -oN eth3-sport25.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -g 22 -oN eth3-sport22.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -g 1723 -oN eth3-sport1723.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sU -P0 -g 500 -oN eth3-sport500.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sU -P0 -g 1701 -oN eth3-sport1701.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
```

```

nmap -n -r -vv -sU -P0 -g 53 -oN eth3-sport53.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sU -P0 -g 514 -oN eth3-sport514.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sU -P0 -g 123 -oN eth3-sport123.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -g 1494 -oN eth3-sport1494.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -g 135 -oN eth3-sport135.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&

```

# Next, we'll scan a broader range of upper ports since we know a lot of  
# them are open. We won't limit this to the internal network, though,  
# since we need to make sure they're closed to places we don't want.

```

nmap -n -r -vv -sS -P0 -p 4000-7000 -oN eth3-dports4k-7k.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&

```

# The last scan we want to do of this subnet will be a slower, more  
# stealthy scan, to see if patience gives us anything extra.  
# We do this by adding the -T Sneaky option, which waits 15 seconds  
# between scans.  
# This will take quite a while to run!

```

nmap -n -r -vv -sS -P0 -T Sneaky -oN eth3-sS-Sneaky.log \
10.0.0.0/24,10.0.1.0/26,5.6.7.0/28 &&

```

# That's it for the eth3 interface (we hope...)

## Internal L2TP Interface Scan Results and Analysis:

No surprises here either. As expected, there's a lot open to the domain controllers and Exchange server on the internal network. This is shown in the analysis below.

## Script for scanning the L2TP VPN's external interface:

This is, naturally, very similar to the previous scripts, only the addresses of the subnets scanned (and the accompanying logfile names, of course) are different. We do want to scan all four subnets from here, though. (We shouldn't be able to get to any of the internal subnets at all.)

# External L2TP Interface Firewall Scanning Script

```

# First, do a standard TCP scan. We use the following switches:
#
# -n This tells nmap not to resolve IP addresses to names;
#   it's a little bit faster, plus we don't need names
#
# -r This tells nmap NOT to randomize port scans; this makes

```

```

# our log review a bit easier
#
# -vv This tells nmap to give us detailed, verbose logs
#
# -sT We're doing a "TCP Connect" scan; very noisy & obvious
#
# -O This tells nmap to attempt OS identification
#
# -P0 We don't need to do Pings; in fact, nmap will tell us the target
# host is unavailable and quit if we don't include this.

# -oN Log in human-readable format in the filename that follows
#
# 10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 - The addresses we're scanning

nmap -n -r -vv -sT -O -P0 -oN eth4-sT.log \
    10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&

# Next, we make the following changes:
#
# -sS We're doing a SYN scan now, not a full connect

nmap -n -r -vv -sS -P0 -oN eth4-sS.log \
    10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&

# Now we try an ACK scan, using the -sA switch. This is a good way to
# see if a firewall is stateful or not.

nmap -n -r -vv -sA -P0 -oN eth4-sA.log \
    10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&

# Perform a FIN scan, using -sF; this is a bit stealthier than -sS

nmap -n -r -vv -sF -P0 -oN eth4-sF.log \
    10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&

# Try a "Christmas Tree" scan; this sends packets with FIN, PSH, and
# URG. Note that this probably won't show different port results than
# the FIN scan, but we want to see how our firewall reacts to it; we're
# expecting to see it on the firewall's logs.

nmap -n -r -vv -sX -P0 -oN eth4-sX.log \
    10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&

# Null scan, using -sN. As with -sX, we're mostly just testing the
# firewall's reaction

nmap -n -r -vv -sN -P0 -oN eth4-sN.log \
    10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&

# IP Protocol scan. Any non-response is viewed as an open port by nmap,
# so this should try to tell us all protocols are allowed...

```



```
nmap -n -r -vv -sO -P0 -oN eth4-sO.log \
10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
```

# Ping scan. This takes the -sP option; sends ACK's as well.

# Note that the -P0 option wouldn't make sense here...

```
nmap -n -r -vv -sP -oN eth4-sP.log \
10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
```

# UDP scan. Unreliable; like -sO, all non-rejects are assumed open.

```
nmap -n -r -vv -sU -P0 -oN eth4-sU.log \
10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
```

# Now that we've completed our "generic" scans, we try some more  
# directed ones.

# The -g option lets us specify a source port for our scans to come from.

# We'll check HTTP, HTTPS, SMTP, SSH, PPTP, ICA, L2TP, DNS, syslog, IKE,  
# and NTP.

# Lastly, note that these are pretending to be replies, so we want -sA  
# or -sU.

```
nmap -n -r -vv -sA -P0 -g 80 -oN eth4-sport80.log \
10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -g 443 -oN eth4-sport443.log \
10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -g 21 -oN eth4-sport21.log \
10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -g 25 -oN eth4-sport25.log \
10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -g 22 -oN eth4-sport22.log \
10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -g 1723 -oN eth4-sport1723.log \
10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sU -P0 -g 500 -oN eth4-sport500.log \
10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sU -P0 -g 1701 -oN eth4-sport1701.log \
10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sU -P0 -g 53 -oN eth4-sport53.log \
10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sU -P0 -g 514 -oN eth4-sport514.log \
10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sU -P0 -g 123 -oN eth4-sport123.log \
10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -g 1494 -oN eth4-sport1494.log \
10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
nmap -n -r -vv -sA -P0 -g 135 -oN eth4-sport135.log \
10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 &&
```

# The last scan we want to do of this subnet will be a slower, more  
# stealthy scan, to see if patience gives us anything extra.

# We do this by adding the -T Sneaky option, which waits 15 seconds  
# between scans.  
# This will take quite a while to run!

```
nmap -n -r -vv -sS -P0 -T Sneaky -oN eth4-sS-Sneaky.log \
10.0.0.0/24,10.0.1.0/26,10.0.2.0/26,5.6.7.0/28 & &
```

# That's it for the eth3 interface (we hope...)

### **External L2TP Interface Scan Results and Analysis:**

No surprises here, either. UDP 500, 1701, and IP Protocol 50 are all we allow; nmap shows all UDP and all protocols as open, since it doesn't receive rejection responses.

**C.** Our functionality check, as stated before, will simply be a matter of trying to use each service on the "live" network. We will do this service by service, following the list we compiled in our audit plan.

1. Citrix NFuse with Citrix Secure Gateway -- the IT Manager simply logged into the NFuse Website over his home cable modem connection to ensure this worked. All published applications launched successfully.
2. The L2TP over IPsec VPN was successfully logged into in the same manner, followed by a successful (albeit painfully slow) Outlook synchronization.
3. SSH connections were successfully established with the SSH server in the DMZ from both outside the office and inside the office; sessions were then launched from there to successfully establish sessions inside the office, to other servers in the DMZ (though obviously not related to firewall testing, it was still important functionality to ensure), to outside the firewall, and to the firewall itself.
4. The [www.giac.com](http://www.giac.com) Website was accessed from inside and outside the firewall.
5. Web sites outside of the office were successfully accessed from the IT staff and other office staff's PC's inside the office.
6. FTP sessions were established successfully from the IT staff's PC's, but not from the other employees' workstations.
7. E-mail was successfully sent and received.
8. The two syslog servers were checked to ensure they contained records from all of the servers, particularly from servers not located on the same subnet. Both servers had the same entries from the router, firewall, and L2TP server.

9. The Snort IDS systems were checked to ensure they were functioning. [Note that this was done on the live network, not the test one which, of course, had no Snort IDS systems.] The rules were tweaked a little bit in the hopes of reducing false positives; we expect these systems to require a lot of care and feeding...

D. Our work was then checked for completeness, to ensure we had done all we intended to do. We had.

### 3. Evaluating the Audit

E. The first, most obvious mistake we made was in our estimate for how long the audit would take. Our scans took much longer than anticipated, particularly the “Sneaky” scans. We should have anticipated that, since we knew we’d be scanning at least 1554 ports (nmap’s default), separated by 15 seconds each. Simple math brings that to a minimum of 23,310 seconds per scan, or 388.5 minutes -- nearly 6 and a half hours per ip address. In other words, we should have realized our scans were going to take several weeks to complete.

Otherwise, things went pretty well.

Our compiled scans are summarized with the following tables. Note that unless otherwise specified, ports are TCP; our scans reported all UDP as open on all addresses, since our firewall prevents the ICMP that would normally notify our scanner that the ports are unreachable. Likewise, nmap treats any IP protocol it doesn’t receive an explicit denial from as open.

External Interface (eth0)		
IP Address	Expected Open	Scan Results
5.6.7.2 (Firewall)	None	All filtered
5.6.7.7 (SNAT)	None	All filtered
5.6.7.8 (L2TP)	UDP 500,1701; IP 50	All UDP, All protocols open <sup>30</sup>
5.6.7.9 (Firewall)	None	All filtered
5.6.7.10 (Mail, Web)	25,80,443	25,80,443
5.6.7.11 (NFuse)	80,443	80,443
5.6.7.12 (CSG, SSH, Syslog, NTP)	22,443; UDP 512, 123	25,443; all UDP open
10.0.0.21 (DNS, DC)	None	All filtered
10.0.1.2 (WWW)	None	All filtered
10.0.2.12 (L2TP)	None	All filtered

Next, we scanned the internal interface (eth1). From here, we’re trying to reach the DMZ, the VPN, and the Internet. We expect openings to the DMZ only; we

<sup>30</sup> Nmap reports non-responses for these as open; our scans therefore reported all UDP and all IP protocols as open

aren't supposed to have any direct access to the Internet, and the VPN should only be allowing replies.

As expected, we were unable to reach the firewall itself or the VPN. We only succeeded in reaching the services on the DMZ that we were supposed to be able to reach.

Internal Interface (eth1)		
IP Address	Expected Open	Scan Results
10.0.0.1	None	All Filtered
10.0.1.1 (Firewall)	None	All Filtered
10.0.1.2 (WWW)	80,443	80,443
10.0.1.3 (DNS)	UDP 53	All UDP open
10.0.1.6 (CSG)	None	All Filtered
10.0.1.7 (Squid Proxy)	21,80,443	21,80,443
10.0.1.8 (NTP, Syslog)	UDP 123	All UDP open
10.0.1.9 (NFuse)	None	All Filtered
10.0.1.11 (SSH)	22	22
10.0.1.15 (SMTP)	25	25
10.0.1.17 (SUS, NAV)	21, 80	21, 80
10.0.1.28 (SQL)	None	All Filtered
10.0.2.1 (Firewall)	None	All Filtered
10.0.2.12 (L2TP)	None	All Filtered
5.6.7.1 (Router)	None	All Filtered

Next, we scanned the DMZ (eth2). We expect a lot more open here.

We were happy to see this also gave us the expected results.

DMZ Interface		
IP Address	Expected Open	Scan Results
10.0.1.1 (Firewall)	22	22
10.0.0.1 (Firewall)	None	All Filtered
10.0.0.8 (Syslog)	UDP 514	All UDP Open
10.0.0.15 (Exchange)	25	25
10.0.0.17 (File server, SUS)	None	All Filtered
10.0.0.21 (DC, DNS)	UDP 53	All UDP Open
10.0.0.22 (DC, NAV)	None	All Filtered
10.0.0.23 (DC)	None	All Filtered
10.0.0.29 (Citrix MetaFrame)	443,1494	443,1494
10.0.0.30 (Citrix MetaFrame)	443,1494	443,1494
10.0.0.32/27 (IT PC's)	None	All Filtered
10.0.0.128/26 (Staff PC's)	None	All Filtered

5.6.7.1 (Router)	21, 22, 25, 80, 443, 1723; UDP 53, 123, 514; IP protocol 47	21, 22, 25, 80, 443, 1723; All UDP Open; All Protocols Open
5.6.7.2 (Firewall)	None	All Filtered
10.0.2.1 (Firewall)	None	All Filtered
10.0.2.12 (L2TP)	UDP 123, 514	All UDP Open

Happy with what we see here (remember, we expect our UDP & protocol scans to report everything open), we move on to a summary of the internal L2TP VPN interface (eth3).

Internal L2TP Interface		
IP Address	Expected Open	Scan Results
10.0.2.1 (Firewall)	None	All Filtered
10.0.0.1 (Firewall)	None	All Filtered
10.0.0.8 (Syslog)	UDP 514	All UDP Open
10.0.0.15 (Exchange)	88, 135, 139, 389, 445, 4444-6, 5555-5655	88, 135, 139, 389, 445, 4444-6, 5555-5655 open
10.0.0.17 (File server, SUS)	None	None
10.0.0.21 (DC, DNS)	88, 135, 139, 389, 445, 5555-5655; UDP 53	88, 135, 139, 389, 445, 5555-5655 open; All UDP Open
10.0.0.22 (DC, NAV)	21, 88, 135, 139, 389, 445, 5555-5655	21, 88, 135, 139, 389, 445, 5555-5655 open
10.0.0.23 (DC)	88, 135, 139, 389, 445, 5555-5655	88, 135, 139, 389, 445, 5555-5655 open
10.0.0.29 (Citrix MetaFrame)	None	All Filtered
10.0.0.30 (Citrix MetaFrame)	None	All Filtered
10.0.0.32/27 (IT PC's)	None	All Filtered
10.0.0.128/26 (Staff PC's)	None	All Filtered
10.0.1.1 (Firewall)	None	All Filtered
10.0.1.2 (WWW)	80,443	All Filtered
10.0.1.3 (DNS)	UDP 53	All UDP open
10.0.1.6 (CSG)	None	All Filtered
10.0.1.7 (Squid Proxy)	21,80,443	All Filtered
10.0.1.8 (NTP, Syslog)	UDP 123	All UDP open
10.0.1.9 (NFuse)	None	All Filtered
10.0.1.11 (SSH)	22	All Filtered
10.0.1.15 (SMTP)	25	All Filtered
10.0.1.17 (SUS, NAV)	None	All Filtered
10.0.1.28 (SQL)	None	All Filtered
5.6.7.12	None	All Filtered
5.6.7.1	None	All Filtered

Lastly, we have the scans of the interface connected to the external side of the L2TP VPN server (eth4).

External L2TP Interface		
IP Address	Expected Open	Scan Results
5.6.7.2 (Firewall)	None	All Filtered
5.6.7.1 (Router)	UDP 500, 1701; IP Protocol 50	All UDP Open; All Protocols Open
10.0.2.1 (Firewall)	None	All Filtered
10.0.0.1 (Firewall)	None	All Filtered
10.0.0.8 (Syslog)	None	All Filtered
10.0.0.15 (Exchange)	None	All Filtered
10.0.0.17 (File server, SUS)	None	All Filtered
10.0.0.21 (DC, DNS)	None	All Filtered
10.0.0.22 (DC, NAV)	None	All Filtered
10.0.0.23 (DC)	None	All Filtered
10.0.0.29 (Citrix MetaFrame)	None	All Filtered
10.0.0.30 (Citrix MetaFrame)	None	All Filtered
10.0.0.32/27 (IT PC's)	None	All Filtered
10.0.0.128/26 (Staff PC's)	None	All Filtered
10.0.1.1 (Firewall)	None	All Filtered
10.0.1.2 (WWW)	None	All Filtered
10.0.1.3 (DNS)	None	All Filtered
10.0.1.6 (CSG)	None	All Filtered
10.0.1.7 (Squid Proxy)	None	All Filtered
10.0.1.8 (NTP, Syslog)	None	All Filtered
10.0.1.9 (NFuse)	None	All Filtered
10.0.1.11 (SSH)	None	All Filtered
10.0.1.15 (SMTP)	None	All Filtered
10.0.1.17 (SUS, NAV)	None	All Filtered
10.0.1.28 (SQL)	None	All Filtered

We see from these scan summaries that no traffic we want to allow is allowed; our firewall is, indeed, effectively filtering our traffic as desired. [At least for TCP connections; as stated previously, UDP and IP protocols are difficult to impossible to accurately prove open or closed due to their connectionless nature.]

This can also be shown with the following color-coded table; while this table does not show ports, it does show which addresses can reach which others. Color code follows.

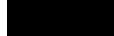



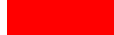

From:	To:	Internet	5.6.7.1	5.6.7.2	5.6.7.7	5.6.7.8	5.6.7.9	5.6.7.10	5.6.7.11	5.6.7.12	10.0.0.8	10.0.0.15	10.0.0.17
Internet													
5.6.7.1 (Router)													
5.6.7.2 (Firewall)													
5.6.7.7 (SNAT)													
5.6.7.8 (L2TP)													
5.6.7.9 (Firewall)													
5.6.7.10 (Mail, Web)													
5.6.7.11 (Ntuse)													
5.6.7.12 (CSG, SSH, Syslog, NTP)													
10.0.0.8 (Syslog, NTP)													
10.0.0.15 (SMTP)													
10.0.0.17 (File server, SUS)													
10.0.0.21 (DC, DNS)													
10.0.0.22 (DC, NAV)													
10.0.0.23 (DC)													
10.0.0.29 (Citrix MetaFrame)													
10.0.0.30 (Citrix MetaFrame)													
10.0.0.32/27 (IT PC's)													
10.0.0.128/26 (Staff PC's)													
10.0.1.1 (Firewall)													
10.0.1.2 (WWW)													
10.0.1.3 (DNS)													
10.0.1.6 (CSG)													
10.0.1.7 (Squid Proxy)													
10.0.1.8 (NTP, Syslog)													
10.0.1.9 (Ntuse)													
10.0.1.11 (SSH)													
10.0.1.15 (SMTP)													
10.0.1.17 (SUS, NAV)													
10.0.1.28 (SQL)													
10.0.2.1 (Firewall)													
10.0.2.12 (L2TP server)													

From:	10.0.0.21	10.0.0.22	10.0.0.23	10.0.0.29	10.0.0.30	10.0.0.32/27	10.0.0.128/26	10.0.1.1	10.0.1.2	10.0.1.3
Internet										
5.6.7.1 (Router)										
5.6.7.2 (Firewall)										
5.6.7.7 (SNAT)										
5.6.7.8 (L2TP)										
5.6.7.9 (Firewall)										
5.6.7.10 (Mail, Web)										
5.6.7.11 (Nfuse)										
5.6.7.12 (CSG, SSH, Syslog, NTP)										
10.0.0.8 (Syslog, NTP)										
10.0.0.15 (SMTP)										
10.0.0.17 (File server, SUS)										
10.0.0.21 (DC, DNS)										
10.0.0.22 (DC, NAV)										
10.0.0.23 (DC)										
10.0.0.29 (Citrix MetaFrame)										
10.0.0.30 (Citrix MetaFrame)										
10.0.0.32/27 (IT PC's)										
10.0.0.128/26 (Staff PC's)										
10.0.1.1 (Firewall)										
10.0.1.2 (WWW)										
10.0.1.3 (DNS)										
10.0.1.6 (CSG)										
10.0.1.7 (Squid Proxy)										
10.0.1.8 (NTP, Syslog)										
10.0.1.9 (Nfuse)										
10.0.1.11 (SSH)										
10.0.1.15 (SMTP)										
10.0.1.17 (SUS, NAV)										
10.0.1.28 (SQL)										
10.0.2.1 (Firewall)										
10.0.2.12 (L2TP server)										



From:	10.0.1.6	10.0.1.7	10.0.1.8	10.0.1.9	10.0.1.11	10.0.1.15	10.0.1.17	10.0.1.28	10.0.2.1	10.0.2.12
Internet										
5.6.7.1 (Router)										
5.6.7.2 (Firewall)										
5.6.7.7 (SNAT)										
5.6.7.8 (L2TP)										
5.6.7.9 (Firewall)										
5.6.7.10 (Mail, Web)										
5.6.7.11 (Nfuse)										
5.6.7.12 (CSG, SSH, Syslog, NTP)										
10.0.8 (Syslog, NTP)										
10.0.15 (SMTP)										
10.0.17 (File server, SUS)										
10.0.21 (DC, DNS)										
10.0.22 (DC, NAV)										
10.0.23 (DC)										
10.0.29 (Citrix MetaFrame)										
10.0.30 (Citrix MetaFrame)										
10.0.32/27 (IT PC's)										
10.0.128/26 (Staff PC's)										
10.0.11 (Firewall)										
10.0.12 (WWW)										
10.0.13 (DNS)										
10.0.16 (CSG)										
10.0.17 (Squid Proxy)										
10.0.18 (NTP, Syslog)										
10.0.19 (Nfuse)										
10.0.111 (SSH)										
10.0.115 (SMTP)										
10.0.117 (SUS, NAV)										
10.0.128 (SQL)										
10.0.21 (Firewall)										
10.0.212 (L2TP server)										

### Color Explanations:

N/A	
Established/Related Only	
New traffic	
Both New and Established	
Filtered	
Same Subnet, not Filtered by Firewall	

Because our firewall is filtering traffic accurately, and we could successfully use all of desired services in part C above, we are happy to report that our firewall has passed our audit.

**F.** This design could be improved upon in a couple of ways, without much expense:

1. Replace the router with a more robust model. The Netopia R5300 is performing adequately, but the cost of a new router (low 4 digits) with more filtering capabilities, more memory, and a more trustworthy operating system is minimal, and would certainly help us sleep better. [As stated earlier, this has been budgeted for 2003.]

2. Eliminate the SSH openings in the firewall. There is truly no reason for this, as IT staff could simply have access to SSH through Citrix NFuse. They could then simply connect to Citrix NFuse, launch SSH, and manage servers as needed in that manner; the only change in process would be the initial connection to Citrix. Note that this also eliminates the need for installing and maintaining SSH on the IT staff's computers (inside the office as well as outside, as they could obviously use Citrix inside the office as well.) [This change is already planned, but because it will be a pilot for the new change management procedures, and because changes to the firewall will require re-doing this audit, closing the SSH holes won't take place until next year.]

3. Eliminate the L2TP over IPsec VPN. This opening in the firewall is also probably unnecessary. Roaming staff could open Outlook via connection to the Citrix NFuse site, then manually copy any new messages (incoming or outgoing). Though tedious, by eliminating the overhead of Outlook itself the data transfer may actually be faster. This would ease ongoing maintenance considerably, and significantly simplify the firewall's rulesets. Plus, of course, there would be one less opening. Changing staff habits, however, will require significant push from upper management; because they don't see a problem with the VPN, this is not expected to change any time soon.

4. Get a larger pipe to the Internet, or a second connection as a backup. GIAC Enterprises' full T-1 line is sufficient for our current needs, but is vulnerable to even simple Denial of Service attacks (see Assignment 4 below.)

5. Perform a full audit, rather than simply focusing on the firewall. GIAC Enterprises has multiple single points of failure; the T-1 mentioned above is only one of them. Ideally, there would be multiple Internet connections, multiple routers, etc. A full audit would help point out those and other weaknesses.

## Assignment 4: Design under fire

We are now going to test the security of a different network. This network is outlined below in *Figure 50*; the details for this network can be found at [http://www.giac.org/practical/Steve\\_Keifling\\_GCFW.doc](http://www.giac.org/practical/Steve_Keifling_GCFW.doc).

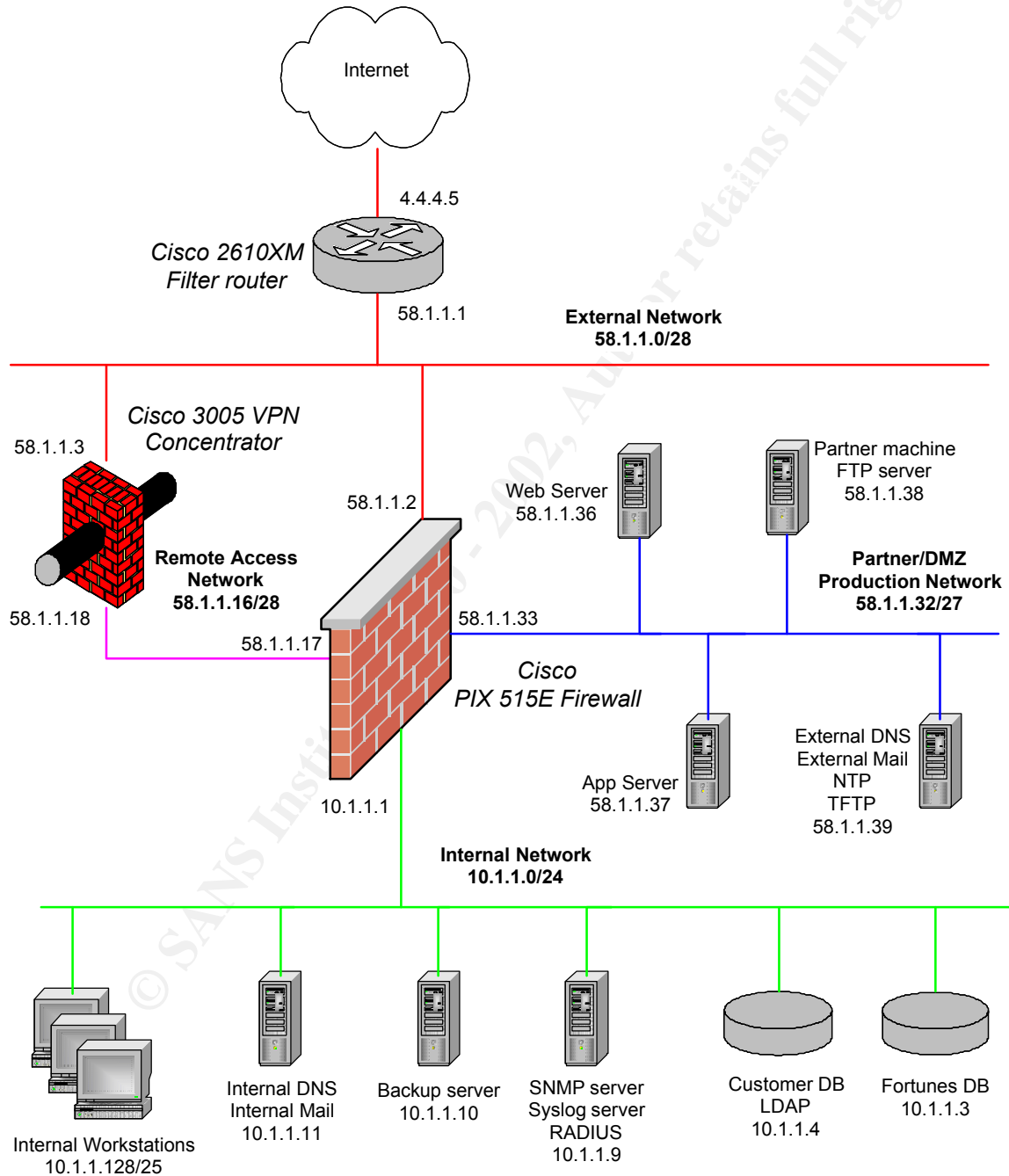


Figure 50

## Overview

There will be three aspects to our attack. First, we will attack the firewall itself. We will then subject the design to a Denial of Service attack. Lastly, we will try to compromise an internal machine through the perimeter.

In the “real world,” all three of our attacks would follow the general outline explained by McClure *et al* in *Hacking Exposed: Network Security Secrets and Solutions, Third Edition*. The first step would be “footprinting,” which entails determining basic information such as the domain names and ip addresses of the network(s) involved, the general layout or structure of the network, etc. The second step is “scanning,” which becomes a bit more risky; because we’re actively sending traffic across the Internet to the target network, our activity may be noticed. This step is required to determine more detailed information, such as which ports are open and which operating systems in use. The lines between this step and the next, which is in-depth “enumeration,” are blurred; once we know what types of systems we’re dealing with, we then choose our targets and try to discern which applications are running, what versions of those applications are in use, etc. We would then research known weaknesses with those applications to determine what sort of attack to implement.

For this exercise, we can skip much of this work; we’ve already been told where the systems are, how the network is laid out, and what applications are running. We are therefore able to focus on the attacks.

### A. Attacking the firewall

The firewall we’ll be attacking is a Cisco PIX 515E, running Operating System Release 6.2 and PIX Device Manager Release 2.0.

We begin by researching vulnerabilities for this firewall. A quick scan on Google<sup>31</sup> brings up a link to the enticing headline, “Cisco PIX Firewall Can Be Crashed By Remote Users Scanning the SSH Port” located on the Security Tracker<sup>32</sup> site at <http://www.securitytracker.com/alerts/2002/Jun/1004643.html>. The title of the article pretty much says it all: assuming the PIX hasn’t been patched since July, taking the firewall off-line should be trivial if they’re using SSH. Interestingly, this weakness is due to a previous patch; this may work to our advantage, as buggy patches can discourage network administrators from applying additional patches in the future. The Security Tracker site conveniently provides a link to Cisco’s own page on the topic, located at <http://www.cisco.com/warp/public/707/SSH-scanning.shtml>.

---

<sup>31</sup> <http://www.google.com>

<sup>32</sup> <http://www.securitytracker.com>

If we send an overly large packet to the SSH port (tcp 22), we will cause the PIX firewall to “consume a large portion of the processor's instruction cycles,<sup>33</sup>” effectively preventing it from doing anything else. This vulnerability may even result in a reboot of the system.

However, further research shows this may not be as trivial as we'd hoped. The headline gives the impression we could just tell nmap to scan port 22 with extra-large packets; however, further reading gives us the impression we have to actually attempt an exploit of the CRC32 exploit in SSH-1. While the original buffer overflow, and the corresponding ability to run code of our own choice, won't succeed, the error appears to show up only during an attempted exploit. The explanation of how to exploit this is at <http://packetstorm.decepticons.org/0102-exploits/ssh1.crc32.txt> -- given our lack of programming prowess, this could get tricky. We aren't giving up yet, though; we'll try to find a pre-written exploit, since there's certainly one or more out there. We did find a lot of interesting stuff at <http://broken.blackroses.com/members/ppp/Hacking.htm> - lots of BackOrifice plugins, for example -- but not what we were actually looking for. We'll move on for now, though -- there's no point killing ourselves trying to write a program to exploit a hole if the whole isn't even available. It's also still possible we could simply send a large packet, as we first thought; in any event, the worst that will happen is our attempts will be noticed. Here's the scan we have in mind:

```
nmap -n -vv -sT -P0 -p 22 --data_length 65535 -oN large-SSH.log 4.4.4.5
```

```
nmap -n -vv -sT -P0 -p 22 --data_length 722 -oN large-SSH.log 4.4.4.5
```

This uses the `--data_length <number>` option to set the packet size in bytes; normal nmap TCP scans send a 40-byte packet. This increases packet size simply by padding with 0's, though, so we clearly can't use this method to exploit buffer overflows to by running code of our choice! We're thinking about trying the 65535 size simply because it's a large number that may make any log reviewers pause in confusion. This will likely be fragmented along the way, however, and we don't know how that would affect our desired outcome, so we'll also try a smaller one as well. Again, note that these are just guesses, but both are large packets for a TCP SYN so if indeed all we really have to do is send an overly large packet to the SSH port, this should do it.

However, the fact that a reboot may be required, or even take place on its own, is enticing enough for us to pause before attempting to exploit this weakness. This is because if we can find another exploit that gives us access to the firewall or otherwise enables us to make changes to the ruleset, we may need a reboot to implement our changes. If we forge ahead and try this first exploit now, we will probably tip our hand, making further compromises much more difficult.

---

<sup>33</sup> <http://www.cisco.com/warp/public/707/SSH-scanning.shtml#details>

So, we go back to our Google search for further research. However, we soon come to the realization that the other PIX vulnerabilities we see have been patched in Operating System Release 6.2. Instead, we focus our search for vulnerabilities to System Release 6.2, and come across <http://www.securiteam.com/securitynews/5FP0P0A7FM.html>, "Weak Cisco PIX Enable Password Encryption Algorithm." This site has a script for computing PIX password hashes, which are based on the MD5 algorithm. So we could, given the chance to log in via telnet or SSH, attempt to brute force the password. The control this would give us is very tempting; we could do a lot more than just crash the system.

Before we try to connect to the firewall, though, we need to check if telnet or SSH are accessible. We do this with nmap:<sup>34</sup>

```
nmap -n -r -vv -sT -P0 -oN pix-check.log 4.4.4.5
```

We find no ports open whatsoever. This leads us to believe that any maintenance must be taking place from inside the network only. This, in turn, means we'd have to compromise an internal machine in order to compromise the firewall; since that's beyond the scope of our assignment, we're out of luck. We don't have anything to run our password brute forcing script against, and there's nothing listening for SSH traffic for us to send a malformed packet to. Our hopes of exploiting either of the two vulnerabilities we've found have been dashed.

### **Result:**

Our attack on the firewall failed.

## ***B. Denial of Service Attack***

We'll try a denial of service attack next. The network we're attacking is defined as being connected to the Internet via a T1, so we're in luck. This should be fairly simple.

Our first step is to find a nice tool to do our dirty work, which will begin by compromising some machines to be our "zombies." We've decided to use Tribe Flood Network 2000 (TFN2K<sup>35</sup>) after a quick glance at the CERT advisories on Denial of Service<sup>36</sup> and Distributed Denial of Service attacks<sup>37</sup>. There are, of course, plenty of other tools available, but we find a certain pleasure in using the same tool that Steve Keifling used in the network we're assaulting.

TFN2K has two parts, a "master" that we operate and a daemon process

<sup>34</sup> <http://www.insecure.org/nmap/>

<sup>35</sup> See <http://online.securityfocus.com/archive/1/49973> for an in-depth overview of this tool

<sup>36</sup> <http://www.cert.org/advisories/CA-1999-17.html>

<sup>37</sup> [http://www.cert.org/incident\\_notes/IN-99-07.html](http://www.cert.org/incident_notes/IN-99-07.html)

running on the agents (a.k.a. “zombies.”)

The first hurdle to overcome is acquiring the tool. (The hacker site we found earlier doesn't have any DoS tools on it at all.) Standard Google searches were not very fruitful; many links we tried going to were no longer active. We eventually found TFN2K at <http://packetstorm.decepticons.org/distributed/> -- note the mis-spelling of the word “deceptions.” We also could have just joined a Usenet group; we wouldn't have to be too particular, just look for one with something along the lines of “hackerz” in the title. It shouldn't take long to have people offer us a copy. No need for that, though, as we've found it.

[Note that the site where we found the tool has a link to the program's “home page” at <http://1337.tsx.org/> -- which says in part,

this site will offer ONLY information and tools to IMPROVE security, NOT TO BREAK IT. Why? First, the distribution of 'malicious' proof-of-concept attacks is unfortunately getting too controversial, and I don't see myself in the duty of getting in the line of fire. Secondly, I'm trying to work with the security community, not against it, and am not willing to provide anything for malicious crackers, institutions, or simply bored teenagers with way too much time on their hands.<sup>38]</sup>

After compiling the program, the next step is to infect at least 50 systems connected to the Internet via cable modem or other fast connection. We do this by comparing the ip addresses of vulnerable computers our randomized nmap scans find with the network assignments of ISP's that provide cable modem and/or DSL service. (We used the whois query at <http://www.geektools.com/cgi-bin/proxy.cgi> to look up a few choice ISP's.)

Here's our nmap scan:

```
nmap -vv -n -sS -O -oA vulnerable_hosts -T Sneaky <target network>
```

Once we had a nice supply of compromised machines, it was time to shut down our targeted network.

We will target the mail server, since port 25 is open. Targeting Web servers on port 80 is probably more common, but we aren't conformists. We will use our zombies to flood the mail server; we don't even really need enough traffic to overwhelm the mail server, though; we just need more than the T1 can handle.

Here is an abbreviated listing of the command options<sup>39</sup>:

TFN2K Client

[-P protocol]

[-S host/ip]

---

<sup>38</sup> Mixter, 2002, <http://1337.tsx.org/>

<sup>39</sup> See [http://newdata.box.sk/2001/oct/TFN2k\\_Analysis-1.3.txt](http://newdata.box.sk/2001/oct/TFN2k_Analysis-1.3.txt) for more details on using the master & client



[-f hostlist]  
[-h hostname]  
[-i target string]  
[-p port]  
<-c command ID>

This is pretty simple. First, we need a listing of our zombies for our script; we put them into a file called “zombiefools.” We then decide how we want our “master” to communicate with the “client” zombies; we’ll use ICMP, so it’s a little more subtle. We want a SYN flood, which is a command ID of 5; we want it to target tcp port 25, the victim’s mail server. We then use this Tribal Flood script:

```
% ./tfn -f zombiefools -P icmp -c 5 -p 25 -i smtp.giac.com
```

This command loads our list of zombie attackers (-f zombiefools), connects to them via ICMP (-P icmp) and tells them to initiate a SYN flood (-c 5) on port 25 (-25) of the target mail server (smtp.giac.com).

The outcome is just a matter of simple math. We’ll use a very conservative estimate for our zombie hosts’ connection speeds, giving them an average of 125Kbps, or 0.125Mbps. We have 50 hosts; 50 times 0.125Mbps comes to 6.25Mbps. The target network is connected to the Internet via a T1, which can handle 1.544Mbps. We’re flooding them with about 4 times the amount of traffic they can handle.

### **Result:**

Success! We’ve taken them off-line.

### **Countermeasure:**

None, short of paying for either an enormous Internet connection, or a second connection as backup. This emphasizes two important things:

- 1) Our own network is just as vulnerable to a distributed Denial of Service attack.
- 2) Personal firewalls for the home user need to become standard. (We’re not optimistic this will happen any time soon...)

## ***C. Compromise a machine through the Perimeter***

Our next task is to attempt to compromise a machine on our target network’s perimeter, in the hopes of using it as a launching pad into the internal network. While our efforts will be focused on the perimeter, our real long-range goal here is the inside.

We have several options for how to go about this. We could, for example, start



by attempting to discover a valid username and password, probably by trying to trick users into visiting a Web site we set up. Never underestimate the value of the phone in such an endeavor, either; particularly with large companies, where the average user may not know all the people in IT, a simple call requesting the user's name & password to "confirm the new equipment will work for you" may be sufficient.

We are in a hurry, however, so we'll start instead by searching for a remotely exploitable vulnerability in one of the machines exposed to inbound traffic from the Internet.

We know from our previous scans that HTTP, HTTPS, SMTP, and FTP are open. We've already used the SMTP server for one attack, and attacking Web servers is almost passé; indeed, we expect the Web server to be the one most watched and patched. So, we'll try to compromise the FTP server instead; it's hopefully a little further down the maintenance chain, so we may have a better chance of catching it unpatched. If we can gain control of that, then we should be able to glean usernames and passwords for use on the inside; since we expect this server to be accessed primarily by IT staff and our victim's partners, we should get some juicy information here.

According to the documentation we have at hand, the FTP server is running wu-ftpd version 2.6.2. So we start with a simple Google search, and quickly come across <http://www.kb.cert.org/vuls/id/886083>, "WU-FTPD does not properly handle file name globbing" and <http://www.kb.cert.org/vuls/id/639760>, "WU-FTPD configured to use RFC 931 authentication running in debug mode contains format string vulnerability." We actually didn't even really need to use Google to find these, as they were both posted on the <http://www.wu-ftp.org> site in November of 2001.<sup>40</sup> Since they're from nearly a year ago, we aren't surprised to see that they've already been patched. We were a little surprised, however, that the link posted there for the Red Hat patch -- <http://www.redhat.com/support/errata/RHSA-2001-147.html> -- is for an lpd vulnerability affecting Red Hat 6.2. Further searching revealed the accurate link, <ftp://updates.redhat.com/7.2/en/os/i386/wu-ftpd-2.6.1-20.i386.rpm> -- but, as the URL indicates, it's for RH version 7.2. RH 7.3, which was released well after this patch, already includes the fix.

We therefore can't take advantage of this vulnerability; it doesn't exist in the server we're targeting. So we keep looking, but fail to find any additional security alerts for this version of ftp.

We then take a moment to search for any vulnerabilities in the Red Hat 7.3 kernel itself. Surprisingly, a search for "Red Hat" at [www.kb.cert.org](http://www.kb.cert.org) (the CERT Coordination Center at the Carnegie Mellon Software Engineering Institute) returned very few results, though it did lead us to

<sup>40</sup> <ftp://ftp.wu-ftp.org/pub/wu-ftpd-attic/cert.org/CA-2001-33>

<http://rhn.redhat.com/errata/RHSA-2002-132.html> -- a vulnerability that can lead to privilege escalation. Naturally, that link includes a link to the patch, which we presume would have been applied already per the state security policy -- but we wouldn't have been able to take advantage of the vulnerability without being able to log on first anyway. We did find listings of a few remotely exploitable vulnerabilities, such as OpenSSL and Ethereal -- but patches for those have been released, and they don't appear to be running anyway.

So we stopped, decided to think what we were missing -- and smacked ourselves in the head. In our zeal to find the latest vulnerability, we skipped over the most obvious, most basic first step in attacking an FTP server: attempting to log on anonymously and see what we can do. Misconfigured security for FTP servers is fairly common; as stated by McClure *et al*,

“Many FTP servers are abused by software pirates who store illegal booty in hidden directories. If your network utilization triples in a day, it might be a good indication that your systems are being used for moving the latest ‘warez.’”<sup>41</sup>

So we went back to basics, and attempted to log in to the FTP server.<sup>42</sup> We simply type: ftp ftp.giac.com and, when prompted for the username, enter anonymous -- we're hoping it will then ask us for our password, which would be our e-mail address by convention (and out of courtesy, with the idea being the server's admin would be able to gauge who is using the server). Of course, there's no cross-checking here, so we could simply enter gbush@badguys.net if we wanted.

We are stymied, however, by the server's lack of anonymous FTP support; our connection is denied. As stated in the paper, “...accounts are tightly controlled and the login records monitored regularly.”<sup>43</sup> This also means that our attempt has probably been logged.

With no way to log into the server, and no remotely exploitable vulnerabilities, our only recourse is to track down a valid username and password. There are numerous ways to do this, varying from the decidedly low-tech (digging through dumpsters) to setting up a mock FTP site, sending an e-mail with a link, and hoping the confused recipients attempt to log in using their valid names and passwords. We currently lack the time or resources for such a detailed endeavor, however.

## Result:

Our attempt to compromise an internal machine via the perimeter failed, because our attempt to compromise the FTP server failed. We could find no

<sup>41</sup> *Hacking Exposed, Third Edition*, Stuart McClure *et al*, p. 333.

<sup>42</sup> There's a nice FTP “HOWTO” at <http://www.ucc.ie/doc/other/howtoftp.html>

<sup>43</sup> [http://www.giac.org/practical/Steve\\_Keifling\\_GCFW.doc](http://www.giac.org/practical/Steve_Keifling_GCFW.doc), page 13.

remotely exploitable vulnerabilities, and our attempts to log onto the server were stymied. Moreover, our attempts were probably noticed.

Given additional time and resources, we could attempt to discover a valid username and password, and then gain access with that.

### **Conclusions:**

The only attack we succeeded at was a Denial of Service attack, which was frighteningly easy. Our other attacks were foiled by effective implementation of security.

This serves as a useful reminder of the importance of what we did in Assignments 1 through 3.

© SANS Institute 2000 - 2002, Author retains full rights.

# Bibliography

## ***Books & other print materials***

Blum, Richard, *Open Source E-mail Security*, Indianapolis: Sams Publishing, Copyright 2002

McClure, Stuart; Scambray, Joel; and Kurtz, George, *Hacking Exposed: Network Security Secrets and Solutions, Third Edition*, Berkeley: The McGraw-Hill Companies, Copyright 2001

Minasi, Mark *et al*, *Mastering Windows 2000 Server, Second Edition*, Alameda: SYBEX Inc., Copyright 1999<sup>44</sup>

Stevens, Richard W., *TCP/IP Illustrated, Volume 1: The Protocols*, Upper Saddle River: Addison-Wesley, Copyright 1994

Zwicky, Elizabeth D; Cooper, Simon; and Chapman, D. Brent, *Building Internet Firewalls, Second Edition*, O'Reilly & Associates, Inc., Sebastopol: Copyright 2000

SANS Institute, "Track 2 - Firewalls, Perimeter Protection and VPN's," Version 2.2, 2002

## ***On-line references***

### **@Stake**

<http://www.atstake.com/research/tools/nc110.tgz> - netcat

### **Beyond Security Lt.**

mao & Thumann, Michael, "Weak Cisco PIX Enable Password Encryption Algorithm," June 21 2002,

<http://www.securiteam.com/securitynews/5FP0P0A7FM.html>

### **blackroses.com**

<http://broken.blackroses.com/members/ppp/Hacking.htm> - listing of hacker tools, with a heavy slant towards BackOrifice and chat. Download & use at your own risk!

### **CD-Linux.org**

2002, <http://cd-linux.org/build-gs.htm> - Project for helping people make CD-

---

<sup>44</sup> Note that Fourth Edition is currently in print

based Linux systems

**Centergate Research Group, LLC**

<http://www.geektools.com/cgi-bin/proxy.cgi> - excellent Web site for "whois" searches, searching multiple servers at once

**CERT/CC**

<http://www.cert.org/> -- Carnegie Mellon University's CERT Coordination Center

<http://www.cert.org/advisories/CA-1999-17.html> - CERT advisory on Denial of Service attacks

[http://www.cert.org/incident\\_notes/IN-99-07.html](http://www.cert.org/incident_notes/IN-99-07.html) - CERT advisory on Distributed Denial of Service attacks

<http://www.kb.cert.org/vuls/id/886083> - "WU-FTPD does not properly handle file name globbing"

<http://www.kb.cert.org/vuls/id/639760> - "WU-FTPD configured to use RFC 931 authentication running in debug mode contains format string vulnerability"

[www.kb.cert.org](http://www.kb.cert.org) - Knowledge Base of the CERT Coordination Center at the Carnegie Mellon Software Engineering Institute

**Cisco Systems, Inc.**

<http://www.cisco.com/warp/public/707/SSH-scanning.shtml> - SSH scanning vulnerability info from Cisco Systems

**Citrix Systems, Inc.**

<http://www.citrix.com/products/metaframexp.asp> - Information on Citrix MetaFrame XP

[http://hgextsrvsft01.citrix.com/cgi-bin/webcgi.exe/?Session=4822373,U=1,ST=187,N=0005,K=1249,SXI=12,Cas e=obj\(13815\)](http://hgextsrvsft01.citrix.com/cgi-bin/webcgi.exe/?Session=4822373,U=1,ST=187,N=0005,K=1249,SXI=12,Cas e=obj(13815)) - How to get Citrix Secure Gateway and Nfuse to run on the same server

[http://hgextsrvsft01.citrix.com/cgi-bin/webcgi.exe/?Session=4892759,U=1,ST=96,N=0005,K=29837,SXI=12,Cas e=obj\(4043\)](http://hgextsrvsft01.citrix.com/cgi-bin/webcgi.exe/?Session=4892759,U=1,ST=96,N=0005,K=29837,SXI=12,Cas e=obj(4043)), "NFuse Classic 1.7 Error: Blank page at redirect.asp after moving web pages"

**Counterpane Internet Security, Inc.**

<http://www.counterpane.com/blowfish.html> - Bruce Schneier's main Blowfish site

<http://www.counterpane.com/pptpv2-paper.html> - "Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2)"

**Elron Software**

<http://www.elronsoftware.com/productfamily/firewall.shtml> - Elron Software's Firewall

<http://www.elronsoftware.com/productfamily/msginspector.shtml> - Elron Software's Message Inspector (e-mail monitoring)

### **Global Information Assurance Certification (GIAC)**

[http://www.giac.org/practical/Steve\\_Keifling\\_GCFW.doc](http://www.giac.org/practical/Steve_Keifling_GCFW.doc) - Steve Keifling's GCFW paper

### **Google**

<http://www.google.com> - excellent Web site for searching

### **Insecure.org**

<http://www.insecure.org/nmap/> - nmap

### **Internet Assigned Numbers Authority (IANA)**

<http://www.iana.org/assignments/ipv4-address-space> - List of current IP address assignments

### **Internet Engineering Task Force (IETF)**

Townsley, W. *et al*, RFC 2661, "Layer 2 Tunneling Protocol" August 1999

<http://www.ietf.org/rfc/rfc2661.txt> -

Mockapetris, P. RFC 1035, "DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION," November 1987 <http://www.ietf.org/rfc/rfc1035.txt> (the RFC for DNS)

### **Linux Online Inc.**

Fawcett, Tom. "The Linux Boot Disk HOWTO," v. 4.5 January 2002

<http://www.linux.org/docs/ldp/howto/Bootdisk-HOWTO/cd-roms.html>

### **Microsoft Corporation**

Microsoft Knowledge Base article Q270836, "XCLN: Exchange 2000 Static Port Mappings" August 7, 2002

<http://support.microsoft.com/support/kb/articles/Q270/8/36.asp>

Microsoft Knowledge Base article Q154596, "HOWTO: Configure RPC Dynamic Port Allocation to Work with Firewall" August 6, 1996

<http://support.microsoft.com/support/kb/articles/Q154/5/96.asp>

Microsoft Knowledge Base article Q298369, "How to Configure a Global Catalog Server to Use a Specific Port When Servicing MAPI Clients" May 7, 2001

<http://support.microsoft.com/support/kb/articles/Q298/3/69.asp>

Microsoft's IIS Lockdown tool,

<http://www.microsoft.com/downloads/release.asp?ReleaseID=33961&area=search&ordinal=2>

Microsoft Knowledge Base article Q238131, "How to Disable Socket Pooling" July 23, 1999

<http://support.microsoft.com/support/kb/articles/Q238/1/31.asp>

## **Minasi.com**

Minasi, Mark, Windows 2000/NT Newsletter archives Issue #16 August 2001, "How I Mostly Avoided Code Red Problems," August 2001  
<http://www.minasi.com/showdoc.asp?docname=nws0108.htm> [NOTE: this site requires free registration]

## **.mixter security**

Mixer, 2002, <http://1337.tsx.org/>

## **National Institute of Standards and Technology (NIST)**

[http://csrc.nist.gov/itsec/guidance\\_W2Kpro.html](http://csrc.nist.gov/itsec/guidance_W2Kpro.html) - The CIS Gold Standard for Windows 2000 Professional

## **Netfilter**

[www.netfilter.org](http://www.netfilter.org) - Home page for Netfilter

<http://www.netfilter.org/documentation/index.html#whatis> - The "What is Netfilter" page

<http://www.netfilter.org/download> - Download site for the latest Netfilter/iptables

Andreasson, Oskar, "Iptables Tutorial 1.1.11," 2001,

<http://www.netfilter.org/documentation/tutorials/blueflux/iptables-tutorial.html#AEN70> - kernel setup

Andreasson, Oskar, "Iptables Tutorial 1.1.11," 2001,

<http://www.netfilter.org/documentation/tutorials/blueflux/iptables-tutorial.html#TABLES> - explanation of tables

<http://www.netfilter.org/documentation/FAQ/netfilter-faq-1.html#ss1.5> -

Information on Netfilter's Patch-O-Matic

[http://www.netfilter.org/documentation/pomlist/pom-](http://www.netfilter.org/documentation/pomlist/pom-submitted.html#ip_conntrack_protocol_destroy)

[submitted.html#ip\\_conntrack\\_protocol\\_destroy](http://www.netfilter.org/documentation/pomlist/pom-submitted.html#ip_conntrack_protocol_destroy) - Details on the

IP\_CONNTRACK\_PROTOCOL\_DESTROY module

[http://www.netfilter.org/documentation/pomlist/pom-](http://www.netfilter.org/documentation/pomlist/pom-submitted.html#ip_conntrack_protocol_unregister)

[submitted.html#ip\\_conntrack\\_protocol\\_unregister](http://www.netfilter.org/documentation/pomlist/pom-submitted.html#ip_conntrack_protocol_unregister) - Details on the

IP\_CONNTRACK\_PROTOCOL\_UNREGISTER module

## **Netopia, Inc.**

Features of the Netopia R5300 router [http://www.netopia.com/en-us/equipment/tech/fmw\\_features.html](http://www.netopia.com/en-us/equipment/tech/fmw_features.html)

Product Overview for the Netopia R5300 router, 2001,

<http://www.netopia.com/equipment/pdf/spec/r5000.pdf>

Netopia R5000 Series Routers, *User Reference Guide*, 2000,

<http://www.netopia.com/equipment/pdf/manuals/r5000/leaseref.pdf>

[http://www.netopia.com/en-us/equipment/purchase/fmw\\_update.html](http://www.netopia.com/en-us/equipment/purchase/fmw_update.html) -

Firmware update for the Netopia R5300

<http://www.netopia.com/en-us/equipment/upgrades/411Feat.html> - List of changes in the latest firmware (4.11) for the R5300



## **New Order**

Barlow, Jason & Thrower, Woody. "TFN2K - An Analysis," Rev. 1.3, March 7, 2000 [http://newdata.box.sk/2001/oct/TFN2k\\_Analysis-1.3.txt](http://newdata.box.sk/2001/oct/TFN2k_Analysis-1.3.txt)

## **OpenBSD**

[www.openssh.org](http://www.openssh.org) - OpenSSH main page

<http://www.openssh.org/portable.html> - This page lists the different "ports" for OpenSSH, as well as mirrors for downloading.

<ftp://ftp.ca.openbsd.org/pub/OpenBSD/OpenSSH/portable/INSTALL> - Installation instructions for OpenSSH

## **Open Source Development Network (OSDN)**

<http://ntsyslog.sourceforge.net/> - NTSyslog, enabling Windows machines to send their event log info to a centralized Syslog server

## **PacketStorm**

<http://packetstorm.decepticons.org/distributed/> - Distributed Denial of Service tools, including Tribal Flood Network 2000.

<http://packetstorm.decepticons.org/0102-exploits/ssh1.crc32.txt> - how to exploit the CRC32 vulnerability in SSH-1

## **Penton Technology Media**

<http://secure.duke.com/nt/SecAdmin/index.cfm?Code=sawi251xna> - Windows & .NET Magazine's Security Advisor newsletter

Kruchov, Andrey, "Secure Web Server Installation on Win2K: Create a bastion host IIS machine," October 2001

<http://www.secadministrator.com/articles/index.cfm?ArticleID=22365&pg=1>

Minasi, Mark, Windows & .NET Magazine, "Roll Out Secure Servers," June 2002, <http://www.winnetmag.com/Articles/Index.cfm?ArticleID=24892>

## **Postfix**

<ftp://ftp.cis.fed.gov/pub/postfix/index.html> - Postfix Version 1.1, Patchlevel 11

## **Red Hat, Inc.**

<http://www.redhat.com/> - Red Hat Linux home page

RHSA-2001:147-09, "Remote exploit possible in lpd," November 8, 2001

<http://www.redhat.com/support/errata/RHSA-2001-147.html>

<ftp://updates.redhat.com/7.2/en/os/i386/wu-ftpd-2.6.1-20.i386.rpm> - FTP patch for Red Hat 7.2

RHSA-2002:132-14, "Updated util-linux package fixes password locking race," July 29, 2002, <http://rhn.redhat.com/errata/RHSA-2002-132.html>

*The Official Red Hat Linux 7.3 Reference Guide*, Red Hat Linux 7.3, 2002

<http://www.redhat.com/docs/manuals/linux/RHL-7.3-Manual/ref-guide/>



### **RSA Security, Inc.**

"RSA Security and Citrix Systems Provide Stronger Security for the Virtual Workplace," April 10, 2002,

[http://www.rsasecurity.com/company/news/releases/pr.asp?doc\\_id=1264](http://www.rsasecurity.com/company/news/releases/pr.asp?doc_id=1264)

"RSA Security Expands Licensing of its E-Security Software to Citrix," May 4, 2000 [http://www.rsasecurity.com/company/news/releases/pr.asp?doc\\_id=180](http://www.rsasecurity.com/company/news/releases/pr.asp?doc_id=180)

### **SecurityFocus Online**

<http://online.securityfocus.com/cgi-bin/sfonline/subscribe.pl> - Bugtraq newsletter  
Barlow, Jason "TFN2K - An Analysis," Rev. 1.3, March 7, 2000

<http://online.securityfocus.com/archive/1/49973>

### **SecurityGlobal.net LLC**

<http://www.securitytracker.com> - nice up-to-date listing of vulnerabilities

"Cisco PIX Firewall Can Be Crashed By Remote Users Scanning the SSH Port,"  
June 27 2002, <http://www.securitytracker.com/alerts/2002/Jun/1004643.html>

### **Snort**

[www.snort.org](http://www.snort.org) - Snort Intrusion Detection System

### **Squid**

<http://www.squid-cache.org/> - Squid proxy

### **SurfControl PLC**

<http://www.surfcontrol.com/products/email/> - SurfControl's E-mail Filter software

### **Symantec Corporation**

<http://enterprisesecurity.symantec.com/content/ProductJump.cfm?Product=155&PID=12930860&EID=0> - Symantec Inc.'s Antivirus Corporate Edition

<http://enterprisesecurity.symantec.com/content/ProductJump.cfm?Product=36&PID=12930860&EID=0> - Symantec Inc.'s Desktop Firewall 2.0

### **SysAdm, Audit, Network, Security Institute (SANS)**

CMP Media, 2002, SANS newsletters

<http://www.sans.org/newlook/digests/SAC.htm>

Otto, Brian C., The Easily Recoverable CD-ROM Booted Linux Internet Server: A How-To," January 21, 2002, <http://rr.sans.org/linux/cdrom.php> (NOTE: This site requires free registration)

### **Tripwire, Inc.**

[www.tripwire.com](http://www.tripwire.com) - Home page for Tripwire software, providing notification if, when, and how files change

**University College, Cork**

"How to Use FTP," <http://www.ucc.ie/doc/other/howtoftp.html>

**WU-FTPD Development Group**

<http://www.wu-ftp.org> - home page for FTP server

© SANS Institute 2000 - 2002, Author retains full rights.