GIAC
CERTIFICATIONS

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 permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Security Essentials: Network, Endpoint, and Cloud (Security 401)"
at http://www.giac.org/registration/gsec

Implementing A Secure Wireless Network

By Thomas Burns

11/26/2003

Abstract

Wireless network and data communications has become increasingly popular in recent years. It is a low cost and convenient means of providing network devices the ability to communicate and share information with each other. Through wireless communications, home users can quickly and easily connect one or more computer devices to the internet and to each other without hiring someone to help them do it; businesses can connect physical sites to one another without the expense of cabling; and laptop users have the ability to use one of the many wireless internet connections that are continually being made available in coffee shops, restaurants and airports. As is frequently the case with 'ease-of-use' networking solutions, however, wireless communications expose users to a multitude of insecurities and vulnerabilities. Because most wireless devices, especially those designed for the home market, are expected to work directly out of the box, built-in wireless security features come disabled on most network access points. Enabling these security features while maintaining wireless connectivity requires knowledge and skill beyond the scope of most home users and many commercial users as well. Unfortunately this results in a growing number of 'live' wireless devices being left wide-open and easily accessed by even the most casual of wireless users scanning the airwaves for access points. "In a wireless network, eavesdropping is easy because wireless communications are not easily confined to a physical area" (Potter & Fleck, 2003, p. 25).

When I was asked by my employer to develop a way for a small group of roaming employees to gain access to the company's network and its resources by means of wireless networking I approached the project as a series of, primarily, network security challenges and, secondarily, network access needs. By assuming that all things wireless are, by their very nature, insecure and vulnerable to exploitation, I was able, during the planning stages of the project, to maintain the narrow focus of keeping the network secure while allowing select users select access to limited resources.

It is understood that this group of roaming users, though small initially, will continue to grow as more and more employees migrate from desktop to laptop computing. It seems imperative to me that as more users connect to the company's network via wireless devices, that that same perspective continue to be maintained to avoid compromising network security.

The company's WAN is, primarily, a frame-relay network connecting 24 locations throughout the state of Wisconsin. It is headquartered in Milwaukee, Wisconsin, but its computing resources, including its gateway are housed in Madison, Wisconsin. It has a third major site in Green Bay, Wisconsin. This wireless project is being piloted in Madison. It is anticipated that additional wireless networks will be installed in Milwaukee and Green Bay within the year.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

The underlying objectives of the project are to mitigate wireless communication vulnerabilities as much as possible, while creating a 'paper trail' of logging that will allow a network administrator to assess continually, the activities of each wireless and wired device and the potential for attack and or abuse. Several technologies were employed to accomplish these objectives:

1. A router to create and segregate trusted and untrusted networks: The fact that this is a pilot and proof-of-concept project, it was necessary to keep the costs of the project to a minimum. It was cost-prohibitive and, as it turns out, unnecessary to purchase a router for this project. The company has a number of workstations that are slated for retirement. I borrowed parts from several of these machines from which I built a working router. This workstation/router is a Digital 3000 with a Pentium 233 MHz processor and 180 MB of RAM. It has two 10/100 network interface cards.
2. An operating system (OS) that allows for a configurable and maintainable level of security: The Linux operating system has so much capacity and potential as an internetworking device that it was not a difficult decision to employ it as the operating system for this router. I have enjoyed the highly configurable environment(s) available through Linux, and chose to use Red Hat Linux 9.0 as the OS upon which to build much of this project.
3. A firewall between the trusted and untrusted networks: Red Hat Linux 9.0 includes an application called Netfilter, also known as iptables, as a loadable kernel module. It is an open-source, no-cost, stateful firewall with an extensible, configurable rules base.
4. Centralized user authentication and administration. The wired network employs RADIUS authentication already through the use of Cisco's ACS server. This product has proven itself to be invaluable as a network management tool and will be employed in several different ways for this project.
5. An access point that can be configured to use a variety of authentication mechanisms, including, at a minimum, Cisco's LEAP authentication, RADIUS authentication, and MAC authentication. In this case we chose the Cisco 1100 Wireless Access Point and installed the latest available software image, IOS version 12.2.(11)JA1.

I. Wireless Security Considerations

As network administrator for a mid-sized company I often travel to company locations throughout the state of Wisconsin. More often than not, before I leave, I plug a wireless network card into my laptop, power it on on the seat beside me, boot into a Windows operating system, and launch Netstumbler (www.netstumbler.com). Netstumbler is a utility that detects broadcasting, wireless access points. War Driving, defined on www.wardriving.com as "Driving around looking for wireless networks" (see http://www.wardriving.com/about.php), is the term frequently used to describe this activity. In the past year, I have seen a notable increase in the number of wireless devices that Netstumbler detects and details for me in its display window. It has reached the point where I can predict with growing certainty where Netstumbler will alert me of the existence of access points along the highway. I have come to learn that

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

residential areas where there is new construction and remote industrial complexes are almost certain to have at least one broadcasting access point. Frequently, it is the residential neighborhoods where multiple access points show up in a cluster of always-on wireless connections. Most of these access points, residential and commercial, do not use WEP (Wireless Equivalent Privacy, see below) encryption in an attempt to secure their network transactions. Most also do not modify the access point's default settings. This means that the same SSIDs (Service Set Identifier, see below) show up time and again. Most access point devices come from the manufacturer configured with a default SSID and a default administrator username and password. (As a matter of fact, I have not come across any major access point manufacturers that send their products out without a pre-configured SSID). Anytime that Netstumbler displays "linksys" as the SSID on a detected access point, chances are very good that it is a Linksys device that owns that SSID, and that the device was not modified in any way before it was deployed. In the case of Linksys devices, the default administrator username and password are both 'admin'. If the device is not re-configured and its defaults changed before being deployed, it would take no time for even the casual war-driver to log into the access point device and modify any of its settings. I was sitting in a hotel recently using its hosted wireless connection and, just out of curiosity, launched a browser session and brought up their Linksys access point's administrative interface. I logged in using admin as the username and admin as the password and immediately had root access to the device. Anyone staying at the hotel could have done the same thing; hijacked the access point while casually waiting for his or her Yahoo inbox to refresh.

Frequently my machine acquires IP addresses from broadcasting access points that have DHCP services running and no MAC Address filtering. I stopped in one small town for coffee one day, parked on the main street and acquired an IP address from the law office across the street. Not only would this have allowed me use of their bandwidth for internet access, it also afforded the possibility that I could compromise data on their network. It was simple to determine that the access point was located in the law office because, in an effort to narrow the search, I launched a protocol analyzer and watched as the only other computer on the network, besides mine, broadcast its Windows NETBIOS name, which happened to be the same name as that painted across the storefront window. In all honesty, I learned what I wanted to know, and turned the protocol analyzer off. My interest is in learning how devices communicate, not in what users communicate to each other.

These experiences as a passive war driver have made me acutely aware of the potential problems and vulnerabilities inherent in opening up our corporate network to wireless communication capability. The following list addresses specific security issues related to wireless data transmission and how they were addressed within this project. Each topic and its related security issues are presented side-by-side with the project implementation decisions that were made after reviewing these considerations. The project was developed and implemented with the understanding that wireless communications explicitly implies vulnerability. Therefore the intent of this project is not to eliminate network vulnerability ("If you want your data to be private, don't transmit it

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

over the air" (Wright, 2002)), but to minimize and mitigate these vulnerabilities while ensuring a healthy balance between the company's needs for wireless network access and the need for a secure environment in which to incorporate wireless communications.

WEP

Considerations:

Wired Equivalent Privacy (WEP) was developed to ensure that data that is transmitted between wireless stations is encrypted for purposes of security. "WEP uses a shared key mechanism with a symmetric cipher called RC4 (Potter & Fleck, 2003, p. 14)". In order to use WEP, both the client machine and the access point must use the shared key and the encryption mechanism must match. "The secret key is used to encrypt packets before they are transmitted, and an integrity check is used to ensure that packets are not modified in transit" (Borisov, Goldberg & Wagner). "In practice, most installations use a single key that is shared between all mobile stations and access points. More sophisticated key management techniques can be used to help defend from the attacks we describe; however, no commercial system we are aware of has mechanisms to support such techniques" (Borisov, Goldberg & Wagner). There are a number of problems with WEP not the least of which is the "overall weakness in the encryption mechanism" (Potter & Fleck, 2003, p. 16). WEP keys are statically entered on access points and client network interface cards (NIC). Many NIC vendors offer the ability to enter up to four different WEP keys. These keys can then be rotated as the WEP key on the access point is changed. This can be very difficult, however, for a network administrator to manage. Besides the fact that all four keys need to be installed statically through the NIC's wireless utility leaving no options for dynamic WEP key assignment, the users must all be informed of WEP key rotation changes. This requires users to develop skills using their wireless NIC utilities; modifying the properties of those utilities; and selecting the new WEP key correctly. It also requires that information about the WEP keys be distributed to the company's help desk personnel; that help desk personnel be familiar with each brand of NIC, the NIC's wireless utility interface, and how to enter and select WEP keys correctly. If the WEP key is not selected correctly, the wireless connection will not be initialized and the user will not gain access to the network.

A number of wireless NICs were tested for this project. Only the Cisco 350 Series card allowed the user to avoid entering static WEP keys. This is a feature of using Cisco wireless hardware with Cisco's LEAP authentication protocol (see below). With LEAP authentication, the Cisco 350 Series NIC uses WEP encryption via dynamic WEP key assignment through access point. This avoids having to enter and manage static WEP keys on individual user devices. Instead, this feature offers the flexibility of entering the WEP key one time on the access point without having to enter it also on client machines.

WEP may not offer protection from an attacker who has the tools available to crack the encryption, but it does offer a deterrent to the casual eavesdropper who may be driving

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

past one of the company's facilities that is using a wireless access point for network access. Netstumbler, for example, displays access points that broadcast and use WEP encryption. The amount of effort and technical skills needed to crack the WEP key may be more than someone sitting in a pickup truck with a cup of coffee and a laptop is willing or interested in putting forth. "Granted, WEP, Wired Equivalent Privacy, is not the best, it can be broken, but, it takes far more effort then clear text flowing through the airwaves available to anyone with a few hundred dollars worth of equipment to pick it up like one might grab police calls with a scanner. If wireless is going to be used, it should at least function in the most secure manner available" (DuFresne).

Project Implementation:

Four WEP keys will be used and will be rotated on a regular basis. WEP is configured to be mandatory and set to 128-bit encryption. This level of encryption is the highest that the access point allows. WEP keys will be posted on the internal Information Systems web site. This site is only accessible via SSL (Secure Sockets Layer), which ensures that data accessed from it is also encrypted. The site is developed and maintained using PHP and Mysql to authenticate users. Only users authorized to view pages that list WEP keys will be allowed access to them. The only Cisco 350 Series wireless NIC that the company owns is in the network administrator's laptop. There is no plan to deploy these specific wireless cards to more users. This means that each user will need to have his/her WEP keys manually entered. The help desk will receive instructions on WEP key rotation. It is also understood that any new wireless cards that are purchased, or laptops that are purchased with new, integrated wireless cards will need to have WEP key rotation procedures documented.

The Service Set Identifier (SSID)

Considerations:

"The SSID is not designed, nor intended for use, as a security mechanism" (Cisco, "*Wireless LAN Security Solution Wireless LAN Security White Paper*"). It is sent in clear text through the access point's beacon messages. Turning off the broadcast beacon on the access point can eliminate casual eavesdropping. It is the broadcast beacon that allows Netstumbler to determine access point location. But there are other wireless tools (Kismet, for example: http://www.kismetwireless.net) that can locate an access point that is not broadcasting. "Once the access point is located, decoding wireless transmissions, and determining the SSID can be done easily with the right tools. In short, a SSID should be viewed more as a network name than a password" (Reid & Seide, 2003, p. 218).

I can frequently determine the location of a detected access point with Netstumbler while driving down the highway at 65 miles per hour simply by looking at the displayed SSID. Many times when the SSID is modified by whomever deployed it it is changed to something company-specific that can found prominently displayed on building signage. It is not uncommon to be driving past a building that reads "Doncaster Manufacturing"

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

and have Netstumbler detect and display an access point with the SSID, DoncasterNet. I have come across a number of these personalized SSIDs including “templeinc” (near a Zor Shrine Temple building); “BergWireless1” (near a building with a sign that reads ‘Berg’ on the front); and “franklinwap” (across from a Ben Franklin store); the locations of which can be determined by simply looking out the window.

The simple process of turning off broadcasting eliminates casual wireless browsing because, on an access point where broadcasting is disabled, the SSID must be known, configured, and selected on the wireless NIC in order for the two devices to communicate. Most NICs allow for the configuration of different profiles on the wireless NIC utility. These profiles can be used and selected based upon which access point the user will be closest to at any given time. “The SSID is a construct that allows logical separation of wireless LANs. In general, a client must be configured with the appropriate SSID to gain access to the wireless LAN. The SSID does not provide any data-privacy functions, nor does it truly authenticate the client to the access point” (Cisco, “*Wireless LAN Security Solution Wireless LAN Security White Paper* ”).

Project Implementation:

The SSID naming scheme is based on an internal naming policy. Wireless client network cards have specific SSIDs entered manually and configured specifically for the access point. The Broadcast Beacon is disabled. This feature was tested several times throughout the project using Netstumbler. While it is true that there are tools that can locate access points that do not have the broadcast beacon enabled, Netstumbler did not detect this particular access point at any time during the project.

LEAP

Considerations:

Extensible Authentication Protocol, EAP, was first developed for PPP connections and is presented in rfc 2284. “These authentication protocols are intended for use primarily by hosts and routers that connect to a PPP network server via switched circuits or dial-up lines, but might be applied to dedicated links as well” (Blunk & Vollbrecht, 1998). LEAP (Lightweight Extensible Authentication Protocol) is Cisco's implementation of EAP. Wireless authentication is based, predominately, on device recognition rather than user authentication. “The user of the device is invisible to the authenticator, and so unauthorized users can access the network simply by gaining access to an authorized device” (Cisco, “*Wireless LAN Security Solution Wireless LAN Security White Paper* ”). Cisco's LEAP “is based on authenticating the user rather than the wireless LAN device. With Cisco LEAP, mutual authentication relies on a shared secret – the user's logon password – which is known by the client and the network and is used to respond to challenges between the user and the Remote Authentication Dial-In User Service (RADIUS) server” (Cisco, “*Wireless LAN Security Solution Wireless LAN Security White Paper* ”). “As with most password-based authentication algorithms, Cisco LEAP is vulnerable to dictionary attacks” (Cisco “*Wireless LAN (WLAN) Dictionary Attack on*

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Cisco LEAP").

I have been using Cisco's ACS server and its remarkably comprehensive functionality for years. "Cisco's Access Control Software (ACS) was developed for both the UNIX and Windows platforms. It is a full fledged TACACS+ and RADIUS server and, beginning with version 2.6a, it handles wireless authentication with EAP-Cisco Wireless and EAP-TLS" (Dismukes, *Wireless Security Blackpaper*). I manage two ACS servers, one that provides backup to the primary. Within this application I can create and manage a variety of network objects including users, user groups, network devices, operating systems, and network protocols, to name some of its many powerful features. The ACS server can use Windows network domain databases; LDAP or other external databases; or its own, internal database to define and manage access control list functionality, user and group logon times; individual commands that users or group members do and do not have the authority to execute from a device's command line; the amount of time each authenticated session is allowed to last per user or group member; and what access level of authority users and group members have on select devices. The ACS server also generates copious logs that are useful for troubleshooting and overall network management.

The ACS server also allows administrative access within the ACS application itself to be designed for an individual or group member. This permits only designated administrators and help desk personnel access to view connection attempts. Unfortunately, this level of control does not extend to individual network devices. In other words, access control can only be placed on the 'Failed Attempts' ACS log, not failed attempts for the wireless device(s) only.

Project Implementation:

LEAP is required for Authentication. The user database is maintained on the Cisco ACS server as an external database (The ACS server negotiates with the Windows Active Directory LDAP database for all wireless user authentication credentials). Consideration was given to creating a separate WirelessUsers group within ACS. It was, however, discounted because there was little to gain by doing so. The users that would belong to such a group are already members of a group that is allowed VPN access to the company's network. Within ACS each user can only belong to one group at a time. Assigning a user that needs VPN access to a wireless users group that, potentially has fewer rights or more limited network access, is not acceptable to management, especially because the initial group of wireless users will be company executives. However, using MAC address authentication (see below) performed within the ACS server, a separate security group can be created that will have strict access limitations set for it.

Access control on the ACS server will not be modified to include other administrators such as help desk personnel. The ACS server logs display information on account names, device addresses, and other sensitive information that warrants limiting access to current administrators.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

MAC Address Authentication

Considerations:

Most major manufacturers of wireless access points (Linksys, Cisco, Netgear, to name a few) use MAC address authentication to validate wireless devices that connect to them. "MAC addresses are sent in the clear as required by the 802.11 specification. As a result, in wireless LANs that use MAC authentication, a network attacker might be able to subvert the MAC authentication process by 'spoofing" a valid MAC address.' Nonetheless, enabling the device for Mac Address authentication includes one more layer of security to eliminate the potential for attack by an unskilled attacker" (Cisco, "*Wireless LAN Security Solution Wireless LAN Security White Paper*").

"Nearly all 802.11 cards in use permit their MAC addresses to be altered, often with full support and drivers from the manufacturer. Using Linux open-source drivers, a user can change their MAC address with the ifconfig tool, or with a short C program calling the ioctl() function with the SIOSIFHWADDR flag. Windows users are commonly permitted to change their MAC address by selecting the properties of their network card drivers in the network control panel applet" (Wright, 2003).

"Separate databases pose administrative problems. Each MAC address table located in individual access points represents a separate database. Although some vendors do provide a means of replicating these tables across a group of access points, this solution is ripe for synchronization and updating problems" (Bruce & Potter, 2003, p. 220).

Netfilter, the firewall that will be used on the project's router, also has capacity to filter on MAC addresses. This allows for a fine degree of control over specific allow, deny, and forward rules for specific devices with the obvious caveat that MAC address authentication is limited in its security scope.

Project Implementation:

MAC Addresses are required for device authentication. The MAC address database is maintained through the Cisco ACS server located on the trusted network. It is not maintained on the access point. Each MAC address is configured as belonging to an ACS internal group. Login times for this group are restricted to select business hours. MAC addresses are also necessary for all DHCP address assignment. Each device is assigned a reserved address via a DHCP server on the inside, trusted network. Because these client devices will be connecting to different company locations and subnets, they need to be provided with an IP address specific to their current location. The DHCP server is running version 3.0pl2 of the ISC's dhcpd product installed on a Red Hat Linux 9.0 platform. It is configured to serve addresses for the untrusted Madison network (10.254.1.0/24). Additional configuration entries that will allow these devices to connect using reserved DHCP address on other subnets will be made when wireless access points and routers are deployed to other company locations.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Open vs. Shared Key Authentication

Considerations:

The Cisco 1100 access point allows for the use of either Shared Key Authentication or Open Authentication. With Open Authentication, the access point allows connectivity to any device that requests it (a request is made through the SSID). Open Authentication is more often the recommended protocol. Both Cisco systems and Microsoft recommend Open Authentication. "Shared key authentication is less secure than open system authentication because it requires the exchange of a secret key that is shared by all wireless access points and clients and therefore is more vulnerable to known-text attacks. In addition, if you select this check box for a wireless network that has multiple wireless access points, clients will lose network connectivity when they travel from one wireless access point to a new wireless access point" (Microsoft, "*Define Active Directory-Based Wireless Network Policies*"). In addition to difficulties inherent in managing shared keys, Cisco contends, "The process of exchanging the challenge text occurs over the wireless link and is vulnerable to a man-in-the-middle attack. An eavesdropper can capture both the plain-text challenge text and the cipher-text response. " (Cisco, "*Wireless LAN Security Wireless LAN Security White Paper* "). "...Using the default OPEN authentication actually reduces the possibility of an unauthorized party discovering your WEP encryption key" (Kennedy, 2003).

Project Implementation:

The access point is configured for Open Authentication. In researching this topic, I found no recommendations for Shared Authentication. The overwhelming general consensus influenced this decision.

Wireless Network Management:

Considerations:

Access Point: There are a number of management protocols available to consider in managing both the access point and the clients that connect to it. Simple Network Management Protocol (SNMP), although useful for gathering data about the access point and its function on a network, allows unencrypted data to be passed from the untrusted to the trusted network. The Cisco 1100 allows for an HTTP interface for configuration, but not an HTTPS interface. Although the access point allows the administrator to choose which port to use for http exchanges, none of the data passed between the Network Administrators browser and the access point is encrypted. The Cisco 1100 (IOS version 12.2.(11)JA1) cannot be configured for a pool of DHCP addresses to be handed out to wireless clients. The Cisco 1100 has one read-only user (username = cisco) configured on the device, with options to add additional users.

Network Security: "Frequently, unauthorized users' intentions are to steal bandwidth

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

rather than view and alter the data passing along the wireless network" (Kennedy, 2003). Users are expected to connect to the device within the building rather than in the parking lot or anywhere outside the physical plant. Broadcasting the signal past the front door of the building will allow potential access to it by unauthorized persons.

Project Implementation:

SNMP is disabled on the access point. HTTP was used for initial configuration purposes while the access point was up but not able to connect to the trusted network. It was allowed to run on an obscure port (in this case 8357) in the process. I admit, this was not the most secure method of configuration, and I take some of the blame for using it. I am surprised that Cisco has not developed a built-in SSL server as it has on other devices that can be configured using different, in this case encrypted, protocols. As a matter of fact, before I modified the HTTP port, I connected to it via the standard HTTP port, port 80. There was no other way, initially to connect to the device, because SSH comes disabled by default. I asked a Cisco technical support representative about this and was told that Cisco is moving towards incorporating this feature into its wireless products, but has not done so yet. I opted to maintain the insecure HHTP connection out of convenience as I developed familiarity with the device. HTTP is disabled on the access point now and SSH is enabled and will be used for all access point management from this point on.

The access point is assigned static IP address rather than configured to pull its address from a DHCP server. Initially, it was assigned an address via DHCP. This is the only way that the access point can be provided with an IP address when first setting it up. There appears to be no reason for the access point to be doing name service lookups, so DNS is disabled on the access point also.

Deleted the default cisco username from the list of local administrators and added a new local user with read/write access. Under the header 'Administrator to be Authenticated by', I selected 'Authentication Server if not found in the Local List.' This will allow administrators to have their credentials checked by the Cisco ACS server while also having a local user available that can administer the device if the ACS is not available to the access point (due to a network outage, for example).

Created a new network group on the ACS server called AccessPoints. Added the Cisco 1100 to the AccessPoints group using its hostname, IP address, and an assigned server key and selected RADIUS(Cisco Aironet) as the authentication mechanism.

Limited the range of the radio signal to prevent the device from being detected in the two company parking lots – one on either side of the building. Relying upon the network card's NIC utilities I could see that the access point can still be detected, however, from the street directly in front of the building. The signal at this point is very weak, and throughput was reduced to 1 Mbps (Megabits per second), but I could not reduce the signal strength further and continue to maintain an 11Mbps data rate within the few offices to which roaming users will have wireless access within the building. I assume also that wireless users will not want to be limited to these few offices, and will pressure management to extend the range to the conference room nearby. I may be able to

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

allow access to the conference room also without increasing the signal strength, but I may need to incorporate an additional antenna to do so. To reduce the signal I set the access point to transmit at 5mW (megawatts) and to permit client power to be limited to 30 mW. The Cisco 1100 Access Point's default values are 100 mW for both settings.

Configured the router's firewall to log copiously all network activity. Also enabled RADIUS accounting to ensure that the Cisco ACS server has records of all traffic sent to and from it.

II. Building and Configuring the Router

Installing the Operating System

It is anticipated that this wireless access system will support from 4 to 10 wireless users initially. I have not yet done any performance benchmarking on the low-end Digital 3000 machine used for the project, but the fact that it has so little capacity relative to our server class machines; that it was ready for retirement, making it several years old; and that it is a machine with evident hardware limitations, it seems reasonable to think it has limited capacity relative to network traffic. As more users are added to the system, it is anticipated that the cost of a server class machine will be justified. After assembling the hardware, Red Hat Linux 9.0 with its stock 2.4.20-8 kernel was installed with minimal packages. I installed it with company administrators' text editors of choice (emacs and pico), the network time protocol (NTP) package, tcpdump to ensure that I had packet capture capability, OpenSSH, and the kernel source packages in the event that I needed to recompile the kernel. Following the install, I saw that I should have stripped the machine down to even fewer packages given that its function is to serve as a router. I used the graphical interface to install the operating system. What I should have done after going through the initial package choice list was to have then selected and de-selected individual packages to ensure that I didn't end up with more applications and utilities than I needed. I inadvertently installed NFS and portmapper, for example, along with the infra-red utilities, the pcmcia kernel package, anacron, and several others.

Regardless, the following packages must be installed:

1. dhcp-3.0pl1-23
2. iptables-1.2.7a-2
3. sendmail-8.12.8-4

This machine will not run a dhcpd server service, but will forward DHCP requests via the dhrelay utility that is included with the dhcp-3.0pl1-23 package. Following installation and successful boot up, I turned off and disabled all the services that were unnecessary to run on the machine including nfs, nfslock and portmap to clean up my installation oversights; anacron, because I will be running cron jobs; rhnsd, because there is no need to contact the Red Hat network for system updates for this machine; apmd, because there is no need for power management; irda; and xinetd, because none of those related services are needed on the box. I also disallowed the iptables and the dhrelay services to be run through standard initialization scripts. Both

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

of these two services will be started through boot up initialization files (see below). When it comes time to rebuild this machine, I will script the rebuild process using kickstart to ensure that the only packages on the machine are those that it needs.

Configuring The Router's Network Settings

Both network cards are assigned static IP addresses:

- Settings for eth0, the internal network card:

```
DEVICE=eth0
BOOTPROTO=static
ONBOOT=yes
NETMASK=255.255.255.0
BROADCAST=10.1.1.255
NETWORK=10.1.1.0
IPADDR=10.1.1.11
```

- Settings for eth1, the external, wireless subnet network card:

```
DEVICE=eth1
BOOTPROTO=static
ONBOOT=yes
NETMASK=255.255.255.0
BROADCAST=10.254.1.255
NETWORK=10.254.1.0
IPADDR=10.254.1.11
```

Each company location is assigned a three-letter code. These codes will be used as machine name prefixes for ease of system administration and log tracking. This machine, located in Madison, was not-too-cleverly named, MSNWirelessRouter. Milwaukee's wireless router will be named MKEWirelessRouter and Green Bay's will be named GRBWirelessRouter. They will not be given static DNS entries. There is no reason for anyone to have to look up the names of these machines to find their IP addresses (security by obscurity). Administrators have access to address information through other means.

Preparing the machine for firewall capability.

Selecting the iptables package at installation ensures that it will be compiled as a kernel module that can be loaded when launched. In this case, it is expected to remain loaded as long as the router is powered on. All iptables logging can be done via syslog and, by default, will log to the /var/log/messages file on the Red Hat machine. I created a separate log for it though to allow for ease of record keeping and management. To do this, I created an iptables directory within the /var/log directory; created a log file called netfilter.log; and edited the /etc/syslog.conf file to log kernel debug messages to that newly created file. I then included a syslog entry to rotate those files daily, and to keep 30 consecutive copies of it before beginning the rotation cycle again. This way, I have a month's worth of log files at my disposal for review. I do not foresee the need to store more than this number of logs on the machine. If I find that it is necessary to maintain more history than that, I will put another script on the device that zips the logs and moves them to another location. The following script details and commands list these procedures:

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Commands/Steps Necessary for Iptables Logging

- `mkdir /var/log/iptables`
- `touch /var/log/iptables/netfilter.log`
- `edit /etc/syslog.conf` and make sure the following line exists:

```
kern.debug   /var/log/iptables/netfilter.log
```

- Add entries in the /etc/logrotate.conf file to rotate logs daily. This can also be added as a separate file with the /etc/logrotate.d/ directory.

```
/var/log/iptables/netfilter.log {
daily
rotate 30
postrotate
            /usr/bin/killall -HUP syslogd
endscript
}
```

e. restart syslog

f. create a cron job that runs logrotate daily. I created a scripts directory in /etc/ that contains a cron file called cronroot. This /etc/scripts directory will hold several other files that are necessary for this project. This is the contents of the /etc/scripts/cronroot file:

```
00 04 * * * /etc/init.d/ntpdate -s -b -p 8 10.1.1.25
#Sets the system clock to the correct time daily

00 00 * * * /usr/sbin/logrotate /etc/logrotate.conf
#Rotates the system logs based on what is in the .conf file
```

g. In order to get the aforementioned ntpdate cron job to work, I modified the /etc/ntp.conf file by adding: server 10.1.1.25 (the internal ntp server). Also added the following line to /etc/ntp/step-tickers: 10.1.1.25. This identifies the ntp server as a trusted device to make significant time adjustments to the machine.

III. Designing and Installing the Iptables Firewall Script

Getting Started

The Netfilter script, when it is completed and ready for execution is named rc.iptables.router and placed in the /etc/rc.d directory. Two entries are then included in the /etc/rc.d/rc.local file, one that executes the firewall script and one that executes the dhrelay command to ensure that DHCP traffic is forwarded to the internal DCHP servers:

```
sh /etc/rc.d/rc.iptables.router
dhcrelay -a 10.1.1.23 10.1.1.24
```

In order to make sure that there is consistency between each wireless router, the script that is used to install the firewall rules has only one section that needs to be modified for each router. Only the sections that are bordered by a string of these characters: (++++) are the sections that require configuration entries. There should be no need to modify any of the rest of the file. Variable assignments are commented to ensure Administrators know the functions and values of those variables. The following script

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

entries were taken from the iptables script and displayed for discussion and clarification purposes. The entire script is too large to print in its entirety, however, a copy of it is attached for reference. It is a shell script that must be made executable and must have as its first line the following:

```
#!/bin/bash
```

The script begins with configuration and installation descriptions for system administrators. It includes much of the information detailed above. The configurable portion of the script begins here:

```
############################
#Section 1.  Machine Functions. Enter 1s (yes) or 0s (no) where needed:
#
#
#+++++++++++++++++++++++++++
# a. Machine name
#This may not need to be changed.  Only fill in the machine's name if it is
#necessary to do so.  Otherwise, leave this as the $HOSTNAME variable.

MACHINENAME=$HOSTNAME

#+++++++++++++++++++++++++++
```

One way to determine whether or not the script has run and the firewall rules are installed is to allow the script to echo its output to the console and to a file. This output is then emailed to an administrator as a status alert on the firewall and its rules.

```
## b. This line sends an email to admins to notify that the script has
# started. At the end of the script is another entry that does a similar
# thing...  but lets the admins know that the script completed successfully.
# If there is no follow-up script... then something occurred to prevent the
# firewall from being installed correctly
#
echo " " > /etc/scripts/NetfilterOutput
echo " "
echo "Iptables firewall script starting . . ."
echo "Iptables firewall script starting . . ." >> /etc/scripts/NetfilterOutput
echo " " > /etc/scripts/NetfilterOutput
echo " "
echo "1. Sending script startup notification email"
echo "1. Sending script startup notification email" >>/etc/scripts/NetfilterOutput
#
mail -s "Netfilter script startup on $MACHINENAME" sysadmin@company.com\
 < /etc/scripts/NetfilterStartup
```

(Another option is to send the email message to root@localhost, then modify the machine's aliases file to send all of root's mail to a specific administrator or a group of administrators. As is typical of the Linux operating system there are many ways to accomplish different tasks.)

```
sleep 1 #This gives the preceding line a chance to execute immediately.
#
# c. path to iptables. This probably doesn't need to be altered:
IPTABLES="/sbin/iptables"
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

The next series of entries allows the script's output to display to standard output within the current user session as well as to write to a file. The file is used to notify administrators that the script has been run and what the status was after running it.

```
#Ok ... let's get this running...
echo "2. Gathering machine details"
echo "2. Gathering machine details" >> /etc/scripts/NetfilterOutput
echo " " >> /etc/scripts/NetfilterOutput
echo "   MachineName: $MACHINENAME" >> /etc/scripts/NetfilterOutput
echo "   MachineName: $MACHINENAME"
```

The next section requires that the MAC addresses of the Access Point (AP) and the wireless devices that will be connecting to it be entered. The firewall will filter on the MAC address, each of which is assigned a reserved IP address by the dhcpd server that resides on the internal network.

```
#++++++++++++++++++++++++++++++
#
# Assign addresses per mac address per machine
#
# First, we need to establish the two networks that we are segmenting
# We will have a Wireless network and a Wired network
        WIRELESS_SUBNET="10.254.1.0/24"
        INTERNAL_SUBNET="10.1.0.0/16"

# Then, enter the information for the Access Point itself
        ACCESSPOINT_IP="10.254.1.200"
        ACCESSPOINT_MAC="00:0d:28:a5:b5:c5"
```

The script is designed to default to five wireless devices. There are subsequent lines for additional devices - up to 10.

```
# If you need to add another machine, do the following:
# 1.  Change the number of mac addresses to the correct number (e.g.
#      NUMBER_OF_MAC_ADDRESSES="7".  THIS CANNOT BE FEWER THAN 5!!")
# 2.  Add another mac address.  Add the appropriate line with the wireless
#      card's mac address and a comment on whose machine it belongs to
# 3.  If the number of mac addresses exceeds 10, the rules will need to be modified.
         NUMBER_OF_MAC_ADDRESSES="5"   #This is limited to 10.  To add more...
                                       #modify the rules. If it is fewer than 5,
                                       #leave it at 5.

        MAC_1="00:90:4b:01:02:03" #Gloern Ulora's Centrino card
        MAC_2="00:90:4b:ab:cd:ef" #Dink Smallwood's Centrino card
        MAC_3="00:0d:bd:1a:1b:1c" #Laura Crofts' Cisco 350 card
        MAC_4="00:0d:65:a1:b2:c3" #Bruce Wayne's Cisco 350 card
        MAC_5="00:90:4b:a2:b2:c2" #Dash Hammett's Centrino card

echo "This is configured for $NUMBER_OF_MAC_ADDRESSES wireless devices only"
echo "This is configured for $NUMBER_OF_MAC_ADDRESSES wireless devices only"\
>> /etc/scripts/NetfilterOutput
```

Network devices are then listed and assigned to individual variables. These devices are fairly static, but because networks and network hosts change, these entries allow for flexibility.

```
#NETWORK settings.
           ADMINS="10.1.1.80/29" #Admin machines 81,82,83,84,85,86
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
DHCP_SERVER_1="10.1.1.23"
DHCP_SERVER_2="10.1.1.24"
PROXY_SERVER="10.1.1.44"
INTRANET_SERVER="10.1.1.45"
IMAP_SERVER="10.1.1.48"
LINUX_HUB="10.1.1.40" #One machine that has allowed access to all others
NTP_SOURCE="10.1.1.25"
SMTP_SERVER="10.1.1.48"
SNMP_SERVER="10.1.1.40"
SYSLOG_SERVER="10.1.1.41"
TACACS_SERVER_1="10.1.1.45"
TACACS_SERVER_2="10.1.1.35"
VAX_SERVER="10.2.1.18"
WIN2K_NTP_SOURCE="10.1.1.39"
LOGIN_SERVER_1="10.1.1.39" # These are all the Win2k Servers.  They are
LOGIN_SERVER_2="10.1.1.35" # all listed in case one is down.
LOGIN_SERVER_3="10.1.1.45"
LOGIN_SERVER_4="10.1.2.35"
LOGIN_SERVER_5="10.1.3.36"
LOGIN_SERVER_6="10.1.4.35"
```

The nameserver variables do not need any modification. The values are assigned dynamically by parsing the /etc/resolv.conf file and selecting the nameserver addresses from it. The /etc/resolv.conf file is, in this case, manually configured so any errors in this file will perpetuate to these variables.

```
            NAMESERVER_1=`/bin/cat /etc/resolv.conf | /bin/grep -m 1 -i "nameserver" |\
/bin/awk '{print $2}'`
            NAMESERVER_2=`/bin/cat /etc/resolv.conf | /bin/grep -i "nameserver" |\
/usr/bin/tail +2 | /bin/awk '{ print $2 }'`
```

For purposes of debugging the script, there is an additional variable named $DEBUGGER that can be assigned either a value of either 1 or 0:

```
#This is for debug purposes.  If you say 1, then you can
# See several values echo'd when you run the script.  Not all values...
# but enough for variable debug purposes

DEBUGGER="0"
```

The $DEBUGGER variable is disabled by default (it is set to 0) but when it is enabled (1) it assigns variables to the following values. These values listed below are then displayed within the user's console session.

```
if [ $DEBUGGER = "1" ]; then
            echo "External interface is  = $EXT_IF"
            echo "Internal interface is  = $INT_IF"
            echo "External address is  = $IPADDR_EXT"
            echo "Internal address is  = $IPADDR_INT"
            echo "nameserver1 is $NAMESERVER_1"
            echo "nameserver2 is $NAMESERVER_2"
            echo "Wireless Subnet is $WIRELESS_SUBNET"
            echo "Internal Subnet is $INTERNAL_SUBNET"
            echo "Access Point address is $ACCESSPOINT_IP"
            echo "Access Point mac address is $ACCESSPOINT_MAC"

fi
```

This ends the portion of the script that is configurable for each machine. The following

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

banner alerts the administrator to that fact. There are configurable entries below this point, but they are within the domain of the firewall administrator, and unnecessary to be considered for script deployment.

```
#++++++++++++++++++++++++++++++
####################################
####################################
####################################
# There is nothing that needs to be
# changed below here!
####################################
####################################
####################################
```

There are two other configurable logging entries. One disables logging that is specifically configured for each firewall rule. Logging is enabled by default. There was a point at which I thought it would be useful to be able to turn off some of the copious logging so that I could peruse more spare log entries. However, I never did find it useful to turn that feature off. Nonetheless, it is still an option, if I need it later.

```
# If you say 1 here you will get every rule (other than kernel hacks and
# standard firewall rules) logging to syslog.  If you say "0", you will get
# generic logging to syslog.  The default is "1".  There is no need to change
# it

LOG_GENERATOR="1"
```

The other entry, $CONNTRACK_LOGGING, assisted in troubleshooting and ensuring that connection tracking, the stateful features of Netfilter, was enabled and working. The amount of logging that this variable enables when set to 1 is too much for general, daily router administration.

```
# If you say "1" here, you will get all the connection tracking logging...
# a lot of stuff. Only say "1" here if you are debugging Connection-tracking
# related traffic
CONNTRACK_LOGGING="0"
```

The script will then grab the machine's network settings and assign values to those variables. As long as the machine is configured properly, the variables assigned will work correctly.

```
#Now let's grab the Eth0 settings

        IPADDR_INT=`/sbin/ifconfig eth0 | /bin/grep "inet addr" |\
 /usr/bin/cut -d':' -f2 | /usr/bin/cut -d ' ' -f1`
        BROADCAST_INT=`/sbin/ifconfig eth0 | /bin/grep "Bcast" |\
 /usr/bin/cut -d':' -f3 | /usr/bin/cut -d ' ' -f1`
        NETMASK_INT=`/sbin/ifconfig eth0 | /bin/grep "Mask" | /usr/bin/cut\
 -d':' -f4 | /usr/bin/cut -d ' ' -f1`

        echo  "IP address, eth0: $IPADDR_INT"
        echo  "IP address, eth0: $IPADDR_INT" >> /etc/scripts/NetfilterOutput
        INT_IF="eth0"

#Now grab the Eth1 settings
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
        IPADDR_EXT=`/sbin/ifconfig eth1 | /bin/grep "inet addr" |\
 /usr/bin/cut -d':' -f2 | /usr/bin/cut -d ' ' -f1`
        BROADCAST_EXT=`/sbin/ifconfig eth1 | /bin/grep "Bcast" |\
 /usr/bin/cut -d':' -f3 | /usr/bin/cut -d ' ' -f1`
        NETMASK_EXT=`/sbin/ifconfig eth1 | /bin/grep "Mask" | /usr/bin/cut\
 -d':' -f4 | /usr/bin/cut -d ' ' -f1`

        echo  "IP address, eth1: $IPADDR_EXT"
        echo  "IP address, eth1: $IPADDR_EXT" >> /etc/scripts/NetfilterOutput
        EXT_IF="eth1"

#And don't forget about the loopback.

        LO_IF="lo"
        LO_IPADDR="127.0.0.1"
```

The script then runs through a series of procedures and rules that allow for tighter security and 'best practices' regarding firewalling. First, it enables 'kernal hacks' by way of echoing 1s (on) to files within the /proc virtual file container; flushes all rules, sets the default policy to drop, and deletes all user-defined chains and tables. It then establishes drop rules based on TCP state flags and ensures that only new connections are made with SYN packets. Then it establishes anti-spoofing rules and creates drop rules for illegal addresses. Several other rules throughout the script allow for specific traffic to and from specific devices including DNS, SNMP, NTP, and Traceroute -related traffic. For details on these rules, the full script is attached.

The following are the ports that were assigned to variables for ease of rule assignment. Assigned to the $LOGIN_PORTS_TCP variable are ports 1026 through 1029. Without these port assignments, client logon was hanging. Tcpdump, the protocol analyzer that I installed with Red Hat 9.0 showed consistently that TCP ports 1026 and 1028 were necessary to complete the logon transactions. Although I did not see TCP ports 1027 and 1029 also needing to be enabled, I came across several internet references that recommended that those ports also be available. In an article on the Gibson Research Corporation site that recommended enabling ports 1025 through 1030 was this editorial reference: “Microsoft operating systems tend to allocate one or more unsuspected, publicly exposed services . . . among the first handful of ports immediately above the end of the service port range (1024+)” (http://www.grc.com/port_1026.htm).

```
##############################
# Port settings ranges, etc.
#
LOGIN_PORTS_UDP="88,389" #These are used for login and LDAP connections
LOGIN_PORTS_TCP="88,389,1026,1027,1028,1029" #without 1026:1029, client logon hangs
PRINTING_PORTS="515,631,9100"
PRIVPORTS="0:1023"
RADIUS_PORTS="1645,1646"
SMB_PORTS_TCP="135,139,445"
SMB_PORTS_UDP="137,138"
SSH_PORTS="1024:65535"
TR_SRC_PORTS="32769:65535" #Traceroute source ports
TR_DEST_PORTS="33434:33523" #Traceroute destination ports
UNPRIVPORTS="1024:65535"
WEB_BROWSER_PORTS="80,443"
####
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

User-Defined Chains

In order to create as fine a level of detail and control over the ruleset and traffic logging as possible, these next entries establish a series of User-Defined chains. The first, prevents DOS (Denial Of Service) attacks by throttling syn connections:

```
######################
## UserDefined Chains.  Creating the Chains:

# UserDefined chain: syn_flood.

#This prevents syn floods: External IP
$IPTABLES -N syn_flood-$EXT_IF
$IPTABLES -A INPUT -i $EXT_IF -p tcp --syn -j syn_flood-$EXT_IF
$IPTABLES -A syn_flood-$EXT_IF -m limit --limit 1/s --limit-burst 4 -j RETURN
$IPTABLES -A syn_flood-$EXT_IF -j DROP
#This prevents syn floods: Internal IP
$IPTABLES -N syn_flood-$INT_IF
$IPTABLES -A INPUT -i $INT_IF -p tcp --syn -j syn_flood-$INT_IF
$IPTABLES -A syn_flood-$INT_IF -m limit --limit 1/s --limit-burst 4 -j RETURN
$IPTABLES -A syn_flood-$INT_IF -j DROP
```

The following are the remaining UserDefined chains:

```
#Now create the rest of the UserDefined chains

        $IPTABLES -N forward_rules
        $IPTABLES -N mac_address
        $IPTABLES -N input_$INT_IF
        $IPTABLES -N input_$EXT_IF
        $IPTABLES -N output_$INT_IF
        $IPTABLES -N output_$EXT_IF
        $IPTABLES -N input_all
        $IPTABLES -N output_all
        $IPTABLES -N icmp_in
        $IPTABLES -N icmp_out
        $IPTABLES -N icmp_fwd
```

Each chain is based on Netfilter's internal filters (FORWARD, INPUT, and OUTPUT). Even though Netfilter allows the use of these internal functions already, by providing a descriptive moniker for each one, it is clearer to the administrator how each rule should read based on traffic flow. This aids in eliminating ambiguity when reading the logs as well as creating new rules.

The rules are then compiled using internal filter functions assigned to individual chains. These assignments are done within the following litany:

```
#Forward rules:
        $IPTABLES -A FORWARD -p ! icmp -j forward_rules
        $IPTABLES -A FORWARD -p ! icmp -j mac_address
        $IPTABLES -A FORWARD -p icmp -j icmp_fwd

#Input rules:
        $IPTABLES -A INPUT -j input_$INT_IF
        $IPTABLES -A INPUT -j input_$EXT_IF
        $IPTABLES -A INPUT -j input_all
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
        $IPTABLES -A INPUT  -p icmp -j icmp_in

#Output rules:
        $IPTABLES -A OUTPUT -j output_$INT_IF
        $IPTABLES -A OUTPUT -j output_$EXT_IF
        $IPTABLES -A OUTPUT -j output_all
        $IPTABLES -A OUTPUT  -p icmp -j icmp_out
```

After defining each chain, all the internal filters should then have drop rules and logging rules associated with them. These drop and log rules are necessary because only references to the internal filters as assigned to the new chains will be allowed from this point onward. Each log rule has the option of appending a descriptive prefix to it. These prefixes are limited to 29 characters, and are very useful when reading through the logs.

```
#Now that the Forward rules are defined, drop all remaining Forward traffic
        $IPTABLES -A FORWARD -j LOG --log-level DEBUG --log-prefix "FWD\
 traffic not defined: "
        $IPTABLES -A FORWARD -j DROP

#Now that the Input rules are defined, drop all remaining Input traffic
#But first we need to drop specific windows-related traffic so we don't fill
#up the logs

        $IPTABLES -A INPUT -p icmp -j LOG --log-level DEBUG –log-prefix\
 "ICMP traffic not defined: "
```

There is no log rule associated with dropping UDP traffic that the internal interface is bombarded with constantly as a result of all the machines on that network that run the Windows operating system. This traffic is simply dropped and not logged.

```
        $IPTABLES -A INPUT -p udp -m multiport --sport $SMB_PORTS_UDP -j DROP
        $IPTABLES -A INPUT -j LOG --log-level DEBUG --log-prefix "INPUT\
 traffic not defined: "

#Now that the Output rules are defined, drop all remaining Output traffic
        $IPTABLES -A OUTPUT -p icmp -j LOG --log-level DEBUG –log-prefix\
 "ICMP traffic not defined: "
        $IPTABLES -A OUTPUT -j LOG --log-level DEBUG --log-prefix "OUTPUT\
 traffic not defined: "
        $IPTABLES -A OUTPUT -j DROP
```

Because this firewall filters on MAC addresses, each MAC address must be entered here and assigned a value. Due to what is anticipated to be low usage initially, it defaults to 5 MAC addresses with capacity for 5 more. It is a simple process to update the script to default, for example, to ten addresses with a capacity for twenty; or whatever is necessary to ensure that there is capacity for however many clients need it. Boosting the maximum capacity above ten immediately requires that this firewall be moved to hardware that can handle that volume of traffic. An example rule and how it fits into the script is shown here. For the remainder of those specific rules, see the attached script.

It may appear that it is an inconsistency in the script to have a configurable section follow an earlier comment that states that nothing needs to be modified below that point.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

However, the rules and access capabilities in this script need to be consistent from one physical location to another. This is not the same as specific router and subnet identities that are necessary for individual machine configuration. Therefore, rules and access changes that are made to one router's script needs to be made to all the company's wireless routers. These changes will be done by the firewall administrator. Other administrators are not expected to configure this portion.

```
######################
## UserDefined Chains: Creating the Rules:  mac_address

        $IPTABLES -A mac_address -i $EXT_IF -m mac –mac-source\
 $ACCESSPOINT_MAC -j ACCEPT
        $IPTABLES -A mac_address -i $EXT_IF -m mac --mac-source $MAC_1\
-j ACCEPT
        $IPTABLES -A mac_address -i $EXT_IF -m mac --mac-source $MAC_2\
-j ACCEPT
        $IPTABLES -A mac_address -i $EXT_IF -m mac --mac-source $MAC_3\
-j ACCEPT
        $IPTABLES -A mac_address -i $EXT_IF -m mac --mac-source $MAC_4\
-j ACCEPT
        $IPTABLES -A mac_address -i $EXT_IF -m mac --mac-source $MAC_5\
-j ACCEPT

if [ $NUMBER_OF_MAC_ADDRESSES -eq 6 ]; then

        $IPTABLES -A mac_address -i $EXT_IF -m mac --mac-source $MAC_6\
-j ACCEPT
        echo "   This is configured for $NUMBER_OF_MAC_ADDRESSES wireless\
devices only"
        echo "   This is configured for $NUMBER_OF_MAC_ADDRESSES wireless\
devices only" >> /etc/scripts/NetfilterOutput
fi
```

There is no need to allow all icmp traffic in and out of the router. Specific ICMP rules are listed to ensure that applications that use ICMP are allowed to function, as well as simple ping requests and echo replies. In order for client machines to communicate with Windows 2000 servers, ICMP type 8 (echo request) continually showed up through iptables logs as a necessary, allowed forward rule. I saw no other traffic that was not already enabled necessary for client connectivity. As more users are added to the system, software applications that use ICMP may show up. The firewall's logging features should display all the information necessary to determine if additional rules are needed.

```
## UserDefined Chains: Creating the Rules: icmp_in; icmp_out
#
#  0: echo reply
#  3: destination unreachable
#  4: source quench
#  8: echo request
# 11: time exceeded
# 12: parameter problem
# 14: timestamp reply

# Accept 3,4,8,11,12 for incoming
$IPTABLES -A icmp_in -p icmp --icmp-type 3 -j ACCEPT
$IPTABLES -A icmp_in -p icmp --icmp-type 4 -j ACCEPT
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
$IPTABLES -A icmp_in -p icmp --icmp-type 8 -j ACCEPT
$IPTABLES -A icmp_in -p icmp --icmp-type 11 -j ACCEPT
$IPTABLES -A icmp_in -p icmp --icmp-type 12 -j ACCEPT

# Allow 0,4,8,11,12,14 only for outgoing icmp.
$IPTABLES -A icmp_out -p icmp --icmp-type 0  -j ACCEPT
$IPTABLES -A icmp_out -p icmp --icmp-type 4  -j ACCEPT
$IPTABLES -A icmp_out -p icmp --icmp-type 8  -j ACCEPT
$IPTABLES -A icmp_out -p icmp --icmp-type 11  -j ACCEPT
$IPTABLES -A icmp_out -p icmp --icmp-type 12  -j ACCEPT
$IPTABLES -A icmp_out -p icmp --icmp-type 14  -j ACCEPT

# Accept type 8 for forwarded icmp. This is necessary for talking to win2k
# servers I haven't found any other icmp types that are necessary as forward
# rules yet....
$IPTABLES -A icmp_fwd -p icmp --icmp-type 8 -j ACCEPT
```

Administrative Rules

There are only a handful of administrators who have rights to manage this server. Their workstations (and one other central Linux server, which is only allowed SSH traffic to the wireless router) are the only ones that are allowed access to the machine. They are also the only ones allowed to manage the wireless devices on the untrusted wireless network.

```
############################
#Specific rules....
#
#
#Admins access
# Allow all traffic inbound from the admins machines and SSH traffic from one
# central linux box only
```

As mentioned earlier, there appears to be no reason to reduce the amount of logging that the firewall does, but I built in the capacity for reducing logging by creating a $LOG_GENERATOR variable. As evidenced in these examples, there are two rule entries associated with each rule. The first is an 'if' statement that checks to see that the LOG_GENERATOR variable exists. If it does, then a log entry will be generated each time traffic passes through the router that corresponds to the rule. Each log entry has a comment appended that describes the rule that applies to the traffic. The second is the rule itself that determines what should be done with that traffic. Generating a log entry with each rule provides administrators a quick and easy way to determine how traffic that passing through the router correlates to firewall rules.

```
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
        $IPTABLES -A input_$INT_IF -i $INT_IF -p all -s $ADMINS  -m state\
 --state NEW,ESTABLISHED -j LOG --log-level DEBUG\
 --log-prefix "in_$INT_IF; src=Admins "
fi

        $IPTABLES -A input_$INT_IF  -i $INT_IF -p all -s $ADMINS  -m state\
 --state  NEW,ESTABLISHED -j ACCEPT

if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
        $IPTABLES -A input_$INT_IF -i $INT_IF -p tcp -s $LINUX_HUB\
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
--dport 22 -m state --state NEW,ESTABLISHED -j LOG --log-level DEBUG\
  --log-prefix "in_$INT_IF: src=LinuxHub. "
fi

        $IPTABLES -A input_$INT_IF  -i $INT_IF -p tcp -s $LINUX_HUB\
--dport 22 -m state --state  NEW,ESTABLISHED -j ACCEPT

if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
        $IPTABLES -A forward_rules -p all -s $ADMINS  -d $WIRELESS_SUBNET\
 -m state --state NEW,ESTABLISHED -j LOG --log-level DEBUG\
 --log-prefix "fw: Admins to Wireless. "
fi

        $IPTABLES -A forward_rules -p all -s $ADMINS -d $WIRELESS_SUBNET\
-m state --state  NEW,ESTABLISHED -j ACCEPT
```

Installing the client access rules:

Rules are subsequently listed that allow the wireless clients to perform the following tasks. See the attached script for details on each of these:

1. Receive DHCP addresses from an internal server.
2. Logon to a Windows 2000 Active Directory enabled network and use network resources such as file and print services and LDAP searches.
3. Telnet to a vax server
4. FTP to an internal server for automated virus definition updates as well as other select downloads
5. Generate external http and https traffic. Send (port 25) and receive (port 110) mail. None of the remote users have their mail agents configured for IMAP traffic. An IMAP server is available to them via their browser, rules for which are enabled via https (port 443).
6. There are also rules for syslog traffic and RADIUS server traffic. RADIUS is used by the Access Point to do Mac and LEAP authentication.

Iptables wrap-up:

Several tasks are listed at the end of the script that accomplish several things:

1. There are final drop rules that ensure that traffic that is not explicitly allowed is explicitly denied. For example, the following rules apply to UDP traffic. Logging rules for UDP traffic are commented out, but available for troubleshooting, if necessary (see attached).

```
$IPTABLES -A input_$EXT_IF  -i $EXT_IF -p udp -j DROP
$IPTABLES -A output_$EXT_IF -o $EXT_IF -p udp -j DROP
$IPTABLES -A input_$INT_IF  -i $INT_IF -p udp -j DROP
$IPTABLES -A output_$INT_IF -o $INT_IF -p udp -j DROP
```

2. There are additional logging rules for all TCP traffic that is attempted but for which there is no allow rule. For example, the following rules apply to TCP traffic.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
# Any tcp not already allowed is logged and then dropped.
$IPTABLES -A input_$EXT_IF  -i $EXT_IF -p tcp -j LOG --log-level DEBUG\
 --log-tcp-options --log-prefix "in_$EXT_IF: TCP "
$IPTABLES -A input_$EXT_IF  -i $EXT_IF -p tcp -j DROP
$IPTABLES -A output_$EXT_IF -o $EXT_IF -p tcp -j LOG --log-level DEBUG\
 --log-tcp-options --log-prefix "out_$EXT_IF: TCP"
$IPTABLES -A output_$EXT_IF -o $EXT_IF -p tcp -j DROP
```

3. IP forwarding is enabled. This should not appear until the end of the script to ensure that no traffic is being forwarded between interfaces before all the rules are defined.

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

4. The file that is generated under /etc/scripts/NetfilterOutput is emailed to the admin who received the first email that notified him/her that the script began.

```
echo " "
echo " . . . Iptables script installed successfully"
echo " "
echo " "  >> /etc/scripts/NetfilterOutput
echo " . . . Iptables script installed successfully" >> /etc/scripts/NetfilterOutput
echo "5. Sending confirmation email . . . Done"
echo "5. Sending confirmation email . . . Done" >> /etc/scripts/NetfilterOutput
echo " "
echo " "  >> /etc/scripts/NetfilterOutput
#
#Now email the output of the script (after it has completed) to an admin for
#service startup verification
#
mail -s "Netfilter script completion and output on $MACHINENAME"\ sysadm@company.com <
/etc/scripts/NetfilterOutput
#
exit 0
```

After loading successfully, the router will now accept, deny, forward, and/or log all traffic on both the untrusted and trusted networks.

Conclusion

Wireless networking and communication is well known within the network security community for its vulnerabilities and insecurities. However, the demand for mobile computing requires that my company institute wireless networking capacity. After having deployed successfully this first stage of the project in Madison, I have come to realize that the use of VPN (Virtual Private Network) connectivity between the untrusted and trusted networks will greatly enhance overall security between devices on these two networks. Doing this will introduce another user authentication mechanism as well as create an encrypted tunnel between the wireless clients and the trusted network. The Cisco ACS server already has a configured, internal group that is allowed VPN access. By modifying the firewall rules, access to the VPN server will be enabled easily. Because the VPN server will then host the wireless client connections, and because the VPN server has its own access control lists and network rights and limitations, I suspect that I can eliminate many of the individual client access rules that I have detailed in the

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Netfilter script. Before completion of this pilot project, VPN access will be incorporated into all allowed traffic and script modifications will be installed. Following the pilot, two more wireless access points will then be deployed in Milwaukee and Green Bay. Additional access points and wireless networks will be deployed as necessary.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

References

Blunk, L. and Vollbrecht, J. "PPP Extensible Authentication Protocol (EAP)" March, 1998. URL:http://www.ietf.org/rfc/rfc2284.txt?number=2284

Borisov, Nikita, Goldberg, Ian, and Wagner, David "(In)Security of the WEP Algorithm" URL:http://www.isaac.cs.berkeley.edu/isaac/wep-faq.html.

Cisco Systems "Cisco Sitewide Tools – Help" URL:http://www.cisco.com/warp/public/779/smbiz/prodconfig/help/eag/ivory/1100/h_ap_sec_ap-client-security.htm

Cisco Systems "Wireless LAN Security Solution Wireless LAN Security White Paper" URL:http://www.cisco.com/en/US/netsol/ns339/ns395/ns176/ns178/networking_solutions_white_paper09186a00800b469f.shtml

Cisco Systems "Wireless LAN (WLAN) Dictionary Attack on Cisco LEAP" August 07, 2003 URL:http://www.cisco.com/en/US/tech/tk722/tk809/technologies_tech_note09186a00801aa80f.shtml

Dismukes, Trey, "Azariah", "Wireless Security Blackpaper" URL: http://arstechnica.com/paedia/w/wireless/security-5.html#Advanced

DuFresne, Ron "Wireless Vendor Woes and Shame" URL: http://sysinfo.com/wired2.html

Kennedy, Susan "Best Practices for Wireless Network Security" November 10, 2003. URL:http://www.computerworld.com/printthis/2003/0,4814,86951,00.html

Microsoft "Define Active Directory-Based Wireless Network Policies" URL:http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/windowsserver2003/proddocs/entserver/add_prefnet_inGP.asp

Potter, Bruce, and Fleck, Bob, 802.11 Security Sebastopol, California: O'Reilly & Associates, 2003

Reid, Neil and Seide, Ron 802.11 (Wi-Fi) Networking Handbook Berkely, California: McGraw-Hill/Osborne, 2003

Wright, Joshua "Detecting Wireless LAN MAC Address Spoofing" January 21, 2003. URL:http://home.jwu.edu/jwright/papers/wlan-mac-spoof.pdf

Wright, Joshua "Security of Wireless Networks" December 18, 2002. URL:http://home.jwu.edu/jwright/wireless.htm

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Appendix

Netfilter firewall script: rc.iptables.router

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
#!/bin/bash
#
#  Iptables firewall script.  For Linux Routers.
#
#  This is divided into several sections ..... for ease of configuring.
#
# 1.  Logging
#           a.  mkdir /var/log/iptables
#           b.  touch /var/log/iptables/netfilter.log
#           c.  edit /etc/syslog.conf and make sure the following line exists:
#                     1. kern.debug               /var/log/iptables/netfilter.log
#
#           d.  Then add entries in the syslog.d/iptables file to rotate logs daily
#
#                     /var/log/iptables/netfilter.log {
#                            daily
#                     rotate 28
#                     postrotate
#                             /usr/bin/killall -HUP syslogd
#                      endscript
#                     }
#
#           e.  restart syslog
#           f.  create a cron job that runs logrotate daily:
#
#                     00 04 * * * /etc/init.d/ntpdate -s -b -p 8 10.1.1.25
#                     #Sets the system clock to the correct time daily
#
#                     00 00 * * * /usr/sbin/logrotate /etc/logrotate.conf
#                     #Rotates the system logs based on what is in the .conf file
#
#           g.  In order to get the aforementioned ntpdate cron job to work, modify
#                     the /etc/ntp.conf file by adding: server 10.1.1.25.  Also add
#                     the following line to /etc/ntp/step-tickers: 10.1.1.25
#
# 2.  Specific Modifications:
#           a.  All the sections that are bordered by a string of these: (++++)
#               are the sections that require configuration entries
#           b.  There should be no need to modify any of the rest of this file
#
# 3.  Installation
#           a.  At the bottom of the file: /etc/rc.d/rc.local add this line:
#           sh /etc/rc.d/rc.iptables.router
#           b.  edit the current /etc/init.d/iptables script. ensure that there is an
#           iptables script variable defined; and the start () function calls it.
#
#                     IPTABLES_SCRIPT="/etc/rc.d/rc.iptables"
#
#
#                                start() {
#
#                                sh /etc/rc.d/rc.iptables.server
#                                exit 0
#                                }
#
#           c. name this script rc.iptables.router and copy it to here:
#           /etc/rc.d/rc.iptables.router
#           d.  This will launch on bootup each time. To start and stop it,
#           just do: #service iptables start .. or .. stop .. or .. restart .. panic ..
#           e.  Install the DHCP-xxx RPM when you build the linux server so that
#           dhcp relays can work.
#           Then...either script it from /etc/init.d or add this line to the rc.local
#           file:
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
#           dhcrelay -a 10.1.1.23 10.1.1.24
#           f. Also check the routing table to be sure that default gateways and
#           routing rules are
#           correct: #route -vn.  If necessary add the necessary routes:
#           (e.g. route add -net 10.1.1.0 netmask 255.255.255.0 dev eth0
#
#
##############################
#Section 1.  Machine Functions. Enter 1s (yes) or 0s (no) where needed:
#
#
#+++++++++++++++++++++++++++++
# a. Machine name
#This may not need to be changed.  Only fill in the machine's name if it is
#necessary to do so.  Otherwise, leave this as the $HOSTNAME variable.

MACHINENAME=$HOSTNAME

#+++++++++++++++++++++++++++++

# b. path to iptables. This probably doesn't need to be altered:
IPTABLES="/sbin/iptables"

#Ok ... let's get this running...
echo " " > /etc/scripts/NetfilterOutput
echo "Iptables firewall script starting . . ." >> /etc/scripts/NetfilterOutput
echo "MachineName: $MACHINENAME" >> /etc/scripts/NetfilterOutput
echo " "
echo "Iptables firewall script starting . . ."
echo "MachineName: $MACHINENAME"

##############################
#Section 2.  Networking and device access
#
#+++++++++++++++++++++++++++++
#
# There are 2 nics in this machine.  eth0 and eth1
# Eth0 will be the internal network, eth1 the wireless, external network
# These addresses should be static, but this script will determine what they
# are and assign values to them
#
# Via eth1 (from the internal network on eth0), dhcp addresses will be handed out
# to all the wireless nics that request one.  These addresses will be from the
# 10.213.x.0/24 network. They are then NAT'd to internal addresses as they get
# forwarded through this linux box.
# Here's what you have to do though:
# Assign addresses per mac address per machine
#
# First, we need to establish the two networks that we are segmenting
# We will have a Wireless network and a Wired network
           WIRELESS_SUBNET="10.254.1.0/24"
           INTERNAL_SUBNET="10.1.0.0/16"


# Then, enter the info for the Access Point itself
           ACCESSPOINT_IP="10.254.1.200"
           ACCESSPOINT_MAC="00:0d:28:a5:b5:c5"

# If you need to add another machine, you need to do the following:
# 1.  Change the number of mac addresses to the correct number (e.g.
#      NUMBER_OF_MAC_ADDRESSES="7".  THIS CANNOT BE FEWER THAN 5!!")
# 2.  Add another mac address.  Add the appropriate line with the wireless
#      card's mac address and a comment on whose machine it belongs to
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
# 3.  If the number of mac addresses exceeds 10 modify the rules.

NUMBER_OF_MAC_ADDRESSES="5" #This is limited to 10.  To add more... modify the rules.
                                        #If it is fewer than 5, leave it at 5.

        MAC_1="00:90:4b:01:02:03" #Gloern Ulora's Centrino card
        MAC_2="00:90:4b:ab:cd:ef" #Dink Smallwood's Centrino card
        MAC_3="00:0d:bd:1a:1b:1c" #Laura Crofts' Cisco 350 card
        MAC_4="00:0d:65:a1:b2:c3" #Bruce Wayne's Cisco 350 card
        MAC_5="00:90:4b:a2:b2:c2" #Dash Hammett's Centrino card

echo "This is configured for $NUMBER_OF_MAC_ADDRESSES wireless devices only"
echo "This is configured for $NUMBER_OF_MAC_ADDRESSES wireless devices only"\
 >> /etc/scripts/NetfilterOutput


#++++++++++++++++++++++++++++++
#

##############################
# Section 3. Server settings.
#

#++++++++++++++++++++++++++++++
#  Specific network values (mail, mgmt, etc.).  These do not need to be modified
#  regularly...
#  They should be, relatively, static
#
#NETWORK settings.
            ADMINS="10.1.1.80/29" #Admin machines 81,82,83,84,85,86
            DHCP_SERVER_1="10.1.1.23"
            DHCP_SERVER_2="10.1.1.24"
            PROXY_SERVER="10.1.1.44"
            INTRANET_SERVER="10.1.1.45"
            IMAP_SERVER="10.1.1.48"
            LINUX_HUB="10.1.1.40" #One machine that has allowed access to all others
            NTP_SOURCE="10.1.1.25"
            SMTP_SERVER="10.1.1.48"
            SNMP_SERVER="10.1.1.40"
            SYSLOG_SERVER="10.1.1.41"
            TACACS_SERVER_1="10.1.1.45"
            TACACS_SERVER_2="10.1.1.35"
            VAX_SERVER="10.2.1.18"
            WIN2K_NTP_SOURCE="10.1.1.39"
            LOGIN_SERVER_1="10.1.1.39" # These are all the Win2k Servers.  They are
            LOGIN_SERVER_2="10.1.1.35" # all listed in case one is down.
            LOGIN_SERVER_3="10.1.1.45"
            LOGIN_SERVER_4="10.1.2.35"
            LOGIN_SERVER_5="10.1.3.36"
            LOGIN_SERVER_6="10.1.4.35"

# The nameservers are gathered dynamically.  There is room for only 2 nameservers.
# To add another, the ruleset must be modified.  (There seems no need to do so.)
# There is also no need to modify the following two entries
            NAMESERVER_1=`/bin/cat /etc/resolv.conf | /bin/grep -m 1 -i "nameserver" |\
 /bin/awk '{print $2}'`
            NAMESERVER_2=`/bin/cat /etc/resolv.conf | /bin/grep -i "nameserver" |\
 /usr/bin/tail +2 | /bin/awk '{ print $2 }'`


#This is for debug purposes.  If you say 1, then you can
# See several values echo'd when you run the script.  Not all values...
# but enough for debug purposes
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
DEBUGGER="0"

#++++++++++++++++++++++++++++++
#
####################################
####################################
####################################
# There is nothing that needs to be
# changed below here!
####################################
####################################
####################################


# If you say 1 here you will get every rule (other than kernel hacks and
# standard firewall rules) logging to syslog.  If you say "0", you will get
# generic logging to syslog.  The default is "1"

LOG_GENERATOR="1"

# If you say "1" here, you will get all the connection tracking logging... a lot of
# stuff
# Only say "1" here if you are debugging Connection-tracking related traffic
CONNTRACK_LOGGING="0"


##############################
# Internet-specific, IPv4 settings
#
#

LOOPBACK="127.0.0.0/8"
CLASS_A="10.0.0.0/8"
CLASS_B="172.16.0.0/12"
CLASS_C="192.168.0.0/16"
CLASS_D_MULTICAST="224.0.0.0/4"
CLASS_E_RESERVED_NET="240.0.0.0/5"
BROADCAST_SRC="0.0.0.0"
BROADCAST_DEST="255.255.255.255"

##############################
# Interfaces (eth0 and eth1) and their values:
#
#
#First, let's grab the Eth0 address

        IPADDR_INT=`/sbin/ifconfig eth0 | /bin/grep "inet addr" | /usr/bin/cut -d':'\
 -f2 | /usr/bin/cut -d ' ' -f1`
        BROADCAST_INT=`/sbin/ifconfig eth0 | /bin/grep "Bcast" | /usr/bin/cut -d':'\
 -f3 | /usr/bin/cut -d ' ' -f1`
        NETMASK_INT=`/sbin/ifconfig eth0 | /bin/grep "Mask" | /usr/bin/cut -d':' -f4\
 | /usr/bin/cut -d ' ' -f1`

            echo  "IP address, eth0: $IPADDR_INT"
            echo  "IP address, eth0: $IPADDR_INT" >> /etc/scripts/NetfilterOutput
            INT_IF="eth0"

#Now grab the Eth1 settings

        IPADDR_EXT=`/sbin/ifconfig eth1 | /bin/grep "inet addr" | /usr/bin/cut -d':'\
 -f2 | /usr/bin/cut -d ' ' -f1`
        BROADCAST_EXT=`/sbin/ifconfig eth1 | /bin/grep "Bcast" | /usr/bin/cut -d':'\
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
 -f3 | /usr/bin/cut -d ' ' -f1`
        NETMASK_EXT=`/sbin/ifconfig eth1 | /bin/grep "Mask" | /usr/bin/cut -d':' -f4\
 | /usr/bin/cut -d ' ' -f1`

        echo  "IP address, eth1: $IPADDR_EXT"
        echo  "IP address, eth1: $IPADDR_EXT" >> /etc/scripts/NetfilterOutput
            EXT_IF="eth1"

            LO_IF="lo"
            LO_IPADDR="127.0.0.1"

##############################
# Port settings ranges, etc.
#
LOGIN_PORTS_UDP="88,389" #These are used for login and LDAP connections
LOGIN_PORTS_TCP="88,389,1026,1027,1028,1029" #without 1026:1029, client logon hangs
PRINTING_PORTS="515,631,9100"
PRIVPORTS="0:1023"
RADIUS_PORTS="1645,1646"
SMB_PORTS_TCP="135,139,445"
SMB_PORTS_UDP="137,138"
SSH_PORTS="1024:65535"
TR_SRC_PORTS="32769:65535" #Traceroute source ports
TR_DEST_PORTS="33434:33523" #Traceroute destination ports
UNPRIVPORTS="1024:65535"
WEB_BROWSER_PORTS="80,443"
####
####
##Debug echoes
# If you are having trouble... Go up to the DEBUGGER entry and
# put in a "1".  It will show you stuff
#to see what your external interface really is
 if [ $DEBUGGER = "1" ]; then
            echo "External interface is  = $EXT_IF"
            echo "Internal interface is  = $INT_IF"
            echo "External address is  = $IPADDR_EXT"
            echo "Internal address is  = $IPADDR_INT"
            echo "nameserver1 is $NAMESERVER_1"
            echo "nameserver2 is $NAMESERVER_2"
            echo "Wireless Subnet is $WIRELESS_SUBNET"
            echo "Internal Subnet is $INTERNAL_SUBNET"
            echo "Access Point address is $ACCESSPOINT_IP"
            echo "Access Point mac address is $ACCESSPOINT_MAC"

fi

#####################################
# General firewall necessities (kernal hacks, startup flushes,
# syn-flood avoidance, TCP flags, icmp rules
#
#
#Kernel hacks

# Enable broadcast echo Protection
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

# Disable Source Routed Packets
echo 1 > /proc/sys/net/ipv4/conf/all/accept_source_route

# Enable TCP SYN Cookie Protection
echo 1 > /proc/sys/net/ipv4/tcp_syncookies

# Disable ICMP Redirect Acceptance
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

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

# Don¹t send Redirect Messags
echo 1 > /proc/sys/net/ipv4/conf/all/send_redirects

# Drop Spoofed Packets coming in on an interface, which if replied to,
# would result in the reply going out a different interface.
echo 1 > /proc/sys/net/ipv4/conf/all/rp_filter

# Log packets with impossible addresses.
echo 1 > /proc/sys/net/ipv4/conf/all/log_martians

# Enable bad error message protection
echo "1" > /proc/sys/net/ipv4/icmp_ignore_bogus_error_responses


##############################################################
#Flush all rules to start with
$IPTABLES -F
$IPTABLES -t nat -F
$IPTABLES -t mangle -F

# Set the default policy to drop
$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

# Unlimited traffic on the loopback interface
$IPTABLES -A INPUT  -i $LO_IF -j ACCEPT
$IPTABLES -A OUTPUT -o $LO_IF -j ACCEPT

#Drop incoming traffic to the loopback from each interface
$IPTABLES -A INPUT  -i $EXT_IF -d $LOOPBACK -j DROP
$IPTABLES -A INPUT  -i $INT_IF -d $LOOPBACK -j DROP

#Same with Broadcast traffic
$IPTABLES -A INPUT  -i $EXT_IF -d $BROADCAST_EXT -j DROP
$IPTABLES -A INPUT  -i $INT_IF -d $BROADCAST_INT -j DROP

$IPTABLES -t nat --policy PREROUTING  DROP
$IPTABLES -t nat --policy OUTPUT DROP
$IPTABLES -t nat --policy POSTROUTING DROP

$IPTABLES -t mangle --policy PREROUTING DROP
$IPTABLES -t mangle --policy OUTPUT DROP

# Remove any pre-existing user-defined chains
$IPTABLES -X
$IPTABLES -t nat -X
$IPTABLES -t mangle -X

##############################################################
# Stealth Scans and TCP State Flags

# All of the bits are cleared
$IPTABLES -A INPUT -p tcp --tcp-flags ALL NONE -j DROP

$IPTABLES -A INPUT -p tcp --tcp-flags SYN,FIN SYN,FIN -j DROP

$IPTABLES -A INPUT -p tcp --tcp-flags SYN,RST SYN,RST -j DROP

# FIN and RST are both set
$IPTABLES -A INPUT -p tcp --tcp-flags FIN,RST FIN,RST -j DROP
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
# FIN is the only bit set, without the expected accompanying ACK
$IPTABLES -A INPUT -p tcp --tcp-flags ACK,FIN FIN -j DROP

# PSH is the only bit set, without the expected accompanying ACK
$IPTABLES -A INPUT -p tcp --tcp-flags ACK,PSH PSH -j DROP

# URG is the only bit set, without the expected accompanying ACK
$IPTABLES -A INPUT -p tcp --tcp-flags ACK,URG URG -j DROP

# Make sure NEW tcp connections are SYN packets
$IPTABLES -A INPUT -i $EXT_IF -p tcp ! --syn -m state --state NEW -j DROP
$IPTABLES -A INPUT -i $INT_IF -p tcp ! --syn -m state --state NEW -j DROP

# User-Defined chain: syn_flood.  This is not listed with the others
# because it doesn't apply its rules to the others

#This prevents syn floods: External IP
$IPTABLES -N syn_flood-$EXT_IF
$IPTABLES -A INPUT -i $EXT_IF -p tcp --syn -j syn_flood-$EXT_IF
$IPTABLES -A syn_flood-$EXT_IF -m limit --limit 1/s --limit-burst 4 -j RETURN
$IPTABLES -A syn_flood-$EXT_IF -j DROP
#This prevents syn floods: Internal IP
$IPTABLES -N syn_flood-$INT_IF
$IPTABLES -A INPUT -i $INT_IF -p tcp --syn -j syn_flood-$INT_IF
$IPTABLES -A syn_flood-$INT_IF -m limit --limit 1/s --limit-burst 4 -j RETURN
$IPTABLES -A syn_flood-$INT_IF -j DROP

#######################

# Refuse spoofed packets pretending to be from
# the machine's own (internal and external) IP addresses
$IPTABLES -A INPUT  -i $EXT_IF -s $IPADDR_EXT -j DROP
$IPTABLES -A INPUT  -i $INT_IF -s $IPADDR_INT -j DROP

# Refuse packets claiming to be from the loopback interface
$IPTABLES -A INPUT  -i $EXT_IF -s $LOOPBACK -j DROP
$IPTABLES -A INPUT  -i $INT_IF -s $LOOPBACK -j DROP

#Refues anything with a Class B address - as defined above
$IPTABLES -A INPUT  -i $EXT_IF -s $CLASS_B -j DROP
$IPTABLES -A INPUT  -i $INT_IF -s $CLASS_B -j DROP

#Refues anything with a Class C address - as defined above
$IPTABLES -A INPUT  -i $EXT_IF -s $CLASS_C -j DROP
$IPTABLES -A INPUT  -i $INT_IF -s $CLASS_C -j DROP

# Refuse Class D multicast addresses illegal as a source address
$IPTABLES -A INPUT -i $EXT_IF -s $CLASS_D_MULTICAST -j DROP
$IPTABLES -A INPUT -i $INT_IF -s $CLASS_D_MULTICAST -j DROP

# Refuse Class E reserved IP addresses
$IPTABLES -A INPUT  -i $EXT_IF -s $CLASS_E_RESERVED_NET -j DROP
$IPTABLES -A INPUT  -i $INT_IF -s $CLASS_E_RESERVED_NET -j DROP


##More Illegal addresses

$IPTABLES -A INPUT -i $EXT_IF -s 0.0.0.0/8 -j DROP
$IPTABLES -A INPUT -i $INT_IF -s 0.0.0.0/8 -j DROP

$IPTABLES -A INPUT -i $EXT_IF -s 169.254.0.0/16 -j DROP
$IPTABLES -A INPUT -i $INT_IF -s 169.254.0.0/16 -j DROP
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
$IPTABLES -A INPUT -i $EXT_IF -s 192.0.2.0/24 -j DROP
$IPTABLES -A INPUT -i $INT_IF -s 192.0.2.0/24 -j DROP

######################
## UserDefined Chains.  Creating the Chains:

            $IPTABLES -N forward_rules
            $IPTABLES -N mac_address
            $IPTABLES -N input_$INT_IF
            $IPTABLES -N input_$EXT_IF
            $IPTABLES -N output_$INT_IF
            $IPTABLES -N output_$EXT_IF
            $IPTABLES -N input_all
            $IPTABLES -N output_all
            $IPTABLES -N icmp_in
            $IPTABLES -N icmp_out
            $IPTABLES -N icmp_fwd

#First, create the rule base.

#Forward rules:
            $IPTABLES -A FORWARD -p ! icmp -j forward_rules
            $IPTABLES -A FORWARD -p ! icmp -j mac_address
            $IPTABLES -A FORWARD -p icmp -j icmp_fwd

#Input rules:
            $IPTABLES -A INPUT -j input_$INT_IF
            $IPTABLES -A INPUT -j input_$EXT_IF
            $IPTABLES -A INPUT -j input_all
            $IPTABLES -A INPUT  -p icmp -j icmp_in

#Output rules:
            $IPTABLES -A OUTPUT -j output_$INT_IF
            $IPTABLES -A OUTPUT -j output_$EXT_IF
            $IPTABLES -A OUTPUT -j output_all
            $IPTABLES -A OUTPUT  -p icmp -j icmp_out

#Now that the Forward rules are defined, drop all remaining Forward traffic
            $IPTABLES -A FORWARD -j LOG --log-level DEBUG --log-prefix\
 "FWD traffic not defined: "
            $IPTABLES -A FORWARD -j DROP

# Now that the Input rules are defined, drop all remaining Input traffic
# But first we need to drop specific windows-related traffic so we don't fill up the
# logs
            $IPTABLES -A INPUT -p icmp -j LOG --log-level DEBUG --log-prefix\
 "ICMP traffic not defined: "
            $IPTABLES -A INPUT -p udp -m multiport --sport $SMB_PORTS_UDP -j DROP
            $IPTABLES -A INPUT -j LOG --log-level DEBUG --log-prefix\
 "INPUT traffic not defined: "
            $IPTABLES -A INPUT -j DROP

#Now that the Output rules are defined, drop all remaining Output traffic
            $IPTABLES -A OUTPUT -p icmp -j LOG --log-level DEBUG --log-prefix\
 "ICMP traffic not defined: "
            $IPTABLES -A OUTPUT -j LOG --log-level DEBUG --log-prefix\
 "OUTPUT traffic not defined: "
            $IPTABLES -A OUTPUT -j DROP
######################

#Connection-Tracking
# This allows all the outgoing/incoming connections to be tracked.
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
# This way you don't need corresponding rules (i.e. for everything
# that goes in you need a corresponding outgoing and vice versa)


if [ $CONNTRACK_LOGGING = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A forward_rules -m state --state ESTABLISHED,RELATED\
 -j LOG --log-level DEBUG --log-prefix "fw: ConnTr. "
fi
            $IPTABLES -A forward_rules -m state --state ESTABLISHED,RELATED -j ACCEPT


if [ $CONNTRACK_LOGGING = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A mac_address -m state --state ESTABLISHED,RELATED\
 -j LOG --log-level DEBUG --log-prefix "mac: ConnTr. "
fi
            $IPTABLES -A mac_address -m state --state ESTABLISHED,RELATED -j ACCEPT

if [ $CONNTRACK_LOGGING = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A input_$INT_IF -s 0/0 -d 0/0 -m state –state\
 ESTABLISHED,RELATED -j LOG --log-level DEBUG --log-prefix "in_$INT_IF: ConnTr. "
            $IPTABLES -A input_$EXT_IF -s 0/0 -d 0/0 -m state –state\
 ESTABLISHED,RELATED -j LOG --log-level DEBUG --log-prefix "in_$EXT_IF: ConnTr. "
            $IPTABLES -A input_all -s 0/0 -d 0/0 -m state --state ESTABLISHED,RELATED\
 -j LOG --log-level DEBUG --log-prefix "in_all: ConnTr. "
fi
            $IPTABLES -A input_$INT_IF -s 0/0 -d 0/0 -m state –state\
 ESTABLISHED,RELATED -j ACCEPT
            $IPTABLES -A input_$EXT_IF -s 0/0 -d 0/0 -m state –state\
 ESTABLISHED,RELATED -j ACCEPT
            $IPTABLES -A input_all -s 0/0 -d 0/0 -m state --state ESTABLISHED,RELATED\
 -j ACCEPT

if [ $CONNTRACK_LOGGING = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A output_$INT_IF -s 0/0 -d 0/0 -m state –state\
 ESTABLISHED,RELATED -j LOG --log-level DEBUG --log-prefix  "out_$INT_IF: ConnTr. "
            $IPTABLES -A output_$EXT_IF -s 0/0 -d 0/0 -m state –state\
 ESTABLISHED,RELATED -j LOG --log-level DEBUG --log-prefix  "out_$EXT_IF: ConnTr. "
            $IPTABLES -A output_all -s 0/0 -d 0/0 -m state --state ESTABLISHED,RELATED\
 -j LOG --log-level DEBUG --log-prefix "out_all: ConnTr. "
fi
            $IPTABLES -A output_$INT_IF -s 0/0 -d 0/0 -m state –state\
 ESTABLISHED,RELATED -j ACCEPT
            $IPTABLES -A output_$EXT_IF -s 0/0 -d 0/0 -m state –state\
 ESTABLISHED,RELATED -j ACCEPT
            $IPTABLES -A output_all -s 0/0 -d 0/0 -m state --state ESTABLISHED,RELATED\
 -j ACCEPT

if [ $CONNTRACK_LOGGING = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A icmp_in -m state --state ESTABLISHED,RELATED\
 -j LOG --log-level DEBUG --log-prefix "icmp_in: ConnTr. "
            $IPTABLES -A icmp_out -m state --state ESTABLISHED,RELATED\
 -j LOG --log-level DEBUG --log-prefix "icmp_out: ConnTr. "
            $IPTABLES -A icmp_fwd -m state --state ESTABLISHED,RELATED\
 -j LOG --log-level DEBUG --log-prefix "icmp_fwd: ConnTr. "
fi
            $IPTABLES -A icmp_in -p icmp -m state --state ESTABLISHED,RELATED -j ACCEPT
            $IPTABLES -A icmp_out -p icmp -m state --state ESTABLISHED,RELATED\
 -j ACCEPT
            $IPTABLES -A icmp_fwd -p icmp -m state --state ESTABLISHED,RELATED\
 -j ACCEPT


#Invalid state
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
#
# This should always have a log rule associated
           $IPTABLES -A input_all -m state --state INVALID -j LOG\
 --log-level DEBUG --log-prefix "input_all: Invalid State. "
           $IPTABLES -A input_all -m state --state INVALID -j DROP
           $IPTABLES -A output_all -m state --state INVALID -j LOG\
 --log-level DEBUG --log-prefix "output_all: Invalid State. "
           $IPTABLES -A output_all -m state --state INVALID -j DROP

######################
## UserDefined Chains: Creating the Rules:  mac_address

           $IPTABLES -A mac_address -i $EXT_IF -m mac --mac-source $ACCESSPOINT_MAC\
 -j ACCEPT
           $IPTABLES -A mac_address -i $EXT_IF -m mac --mac-source $MAC_1 -j ACCEPT
           $IPTABLES -A mac_address -i $EXT_IF -m mac --mac-source $MAC_2 -j ACCEPT
           $IPTABLES -A mac_address -i $EXT_IF -m mac --mac-source $MAC_3 -j ACCEPT
           $IPTABLES -A mac_address -i $EXT_IF -m mac --mac-source $MAC_4 -j ACCEPT
           $IPTABLES -A mac_address -i $EXT_IF -m mac --mac-source $MAC_5 -j ACCEPT

if [ $NUMBER_OF_MAC_ADDRESSES -eq 6 ]; then

           $IPTABLES -A mac_address -i $EXT_IF -m mac --mac-source $MAC_6 -j ACCEPT
           echo "This is configured for $NUMBER_OF_MAC_ADDRESSES\
 wireless devices only"
           echo "This is configured for $NUMBER_OF_MAC_ADDRESSES\
 wireless devices only" >> /etc/scripts/NetfilterOutput
fi
if [ $NUMBER_OF_MAC_ADDRESSES -eq 7 ]; then

           $IPTABLES -A mac_address -i $EXT_IF -m mac --mac-source $MAC_7 -j ACCEPT
           echo "This is configured for $NUMBER_OF_MAC_ADDRESSES\
 wireless devices only"
           echo "This is configured for $NUMBER_OF_MAC_ADDRESSES\
 wireless devices only" >> /etc/scripts/NetfilterOutput
fi
if [ $NUMBER_OF_MAC_ADDRESSES -eq 8 ]; then

           $IPTABLES -A mac_address -i $EXT_IF -m mac --mac-source $MAC_8 -j ACCEPT
           echo "This is configured for $NUMBER_OF_MAC_ADDRESSES\
 wireless devices only"
           echo "This is configured for $NUMBER_OF_MAC_ADDRESSES\
 wireless devices only" >> /etc/scripts/NetfilterOutput
fi
if [ $NUMBER_OF_MAC_ADDRESSES -eq 9 ]; then

           $IPTABLES -A mac_address -i $EXT_IF -m mac --mac-source $MAC_9 -j ACCEPT
           echo "This is configured for $NUMBER_OF_MAC_ADDRESSES\
 wireless devices only"
           echo "This is configured for $NUMBER_OF_MAC_ADDRESSES\
 wireless devices only" >> /etc/scripts/NetfilterOutput
fi
if [ $NUMBER_OF_MAC_ADDRESSES -eq 10 ]; then

           $IPTABLES -A mac_address -i $EXT_IF -m mac --mac-source $MAC_10 -j ACCEPT
           echo "This is configured for $NUMBER_OF_MAC_ADDRESSES\
 wireless devices only"
           echo "This is configured for $NUMBER_OF_MAC_ADDRESSES\
 wireless devices only" >> /etc/scripts/NetfilterOutput
fi


## UserDefined Chains: Creating the Rules: icmp_in; icmp_out; icmp_fwd
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
#
#  0: echo reply (pong)
#  3: destination-unreachable (port-unreachable, fragmentation-needed etc).
#  4: source quench
#  8: echo request (ping)
# 11: time-exceeded
# 12: parameter-problem
# 14: timestamp reply

# Accept 3,4,8,11,12 for incoming
# For debugging ICMP:
#$IPTABLES -A icmp_in -p icmp -j ACCEPT
$IPTABLES -A icmp_in -p icmp --icmp-type 3 -j ACCEPT
$IPTABLES -A icmp_in -p icmp --icmp-type 4 -j ACCEPT
$IPTABLES -A icmp_in -p icmp --icmp-type 8 -j ACCEPT
$IPTABLES -A icmp_in -p icmp --icmp-type 11 -j ACCEPT
$IPTABLES -A icmp_in -p icmp --icmp-type 12 -j ACCEPT

# Allow 0,4,8,11,12,14 only for outgoing icmp.
# Uncomment for debugging ICMP:
#$IPTABLES -A icmp_out -p icmp -j ACCEPT
$IPTABLES -A icmp_out -p icmp --icmp-type 0  -j ACCEPT
$IPTABLES -A icmp_out -p icmp --icmp-type 4  -j ACCEPT
$IPTABLES -A icmp_out -p icmp --icmp-type 8  -j ACCEPT
$IPTABLES -A icmp_out -p icmp --icmp-type 11  -j ACCEPT
$IPTABLES -A icmp_out -p icmp --icmp-type 12  -j ACCEPT
$IPTABLES -A icmp_out -p icmp --icmp-type 14  -j ACCEPT

# Accept type 8 for forwarded icmp. this is necessary for talking to win2k servers
# I haven't found any other icmp types that are necessary for forward rules yet....
# For debugging ICMP:
#$IPTABLES -A icmp_fwd -p icmp -j ACCEPT
$IPTABLES -A icmp_fwd -p icmp --icmp-type 8 -j ACCEPT

##############################
#Specific rules....
#
#
#Admins access
# Allow ssh inbound from the admins machines only (this includes one other server:
# 10.1.1.40).

if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
           $IPTABLES -A input_$INT_IF -i $INT_IF -p all -s $ADMINS  -m state\
 --state NEW,ESTABLISHED -j LOG --log-level DEBUG\
 --log-prefix "in_$INT_IF; src=Admins "
fi

        $IPTABLES -A input_$INT_IF  -i $INT_IF -p all -s $ADMINS  -m state\
 --state  NEW,ESTABLISHED -j ACCEPT

if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
           $IPTABLES -A input_$INT_IF -i $INT_IF -p tcp -s $LINUX_HUB --dport 22\
  -m state --state NEW,ESTABLISHED -j LOG --log-level DEBUG --log-prefix\
 "in_$INT_IF: src=LinuxHub. "
fi

        $IPTABLES -A input_$INT_IF  -i $INT_IF -p tcp -s $LINUX_HUB --dport 22\
  -m state --state  NEW,ESTABLISHED -j ACCEPT

if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
           $IPTABLES -A forward_rules -p all -s $ADMINS  -d $WIRELESS_SUBNET -m state\
 --state NEW,ESTABLISHED -j LOG --log-level DEBUG --log-prefix\
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
 "fw: Admins to Wireless. "
fi

        $IPTABLES -A forward_rules -p all -s $ADMINS -d $WIRELESS_SUBNET\
 -m state --state  NEW,ESTABLISHED -j ACCEPT

## AUTH server
# Reject ident probes with a tcp reset.
# Might need to do this if a mailhost  won't accept mail with a drop only. Log this
# traffic to find out what's up.
           $IPTABLES -A input_$INT_IF -p tcp --dport 113 -j LOG --log-level DEBUG\
 --log-prefix "in_$INT_IF: Port=113. "
           $IPTABLES -A input_$INT_IF -p tcp --dport 113 -j REJECT\
 --reject-with tcp-reset

## TRACEROUTE
# Outgoing traceroute anywhere.
# The reply to a traceroute is an icmp time-exceeded. No need to log this traffic.
           $IPTABLES -A output_$INT_IF -o $INT_IF -p udp --sport $TR_SRC_PORTS\
 --dport $TR_DEST_PORTS -m state --state NEW -j ACCEPT
           $IPTABLES -A output_$EXT_IF -o $EXT_IF -p udp --sport $TR_SRC_PORTS\
 --dport $TR_DEST_PORTS -m state --state NEW -j ACCEPT

##NTP
#
#This rule allows the linux box itself to get ntp settings:
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
           $IPTABLES -A output_$INT_IF -o $INT_IF -p udp -d $NTP_SOURCE  --dport 123\
 -m state --state NEW,ESTABLISHED -j LOG --log-level DEBUG --log-prefix\
 "out_$INT_IF to Ntp. "
fi
           $IPTABLES -A output_$INT_IF -o $INT_IF -p udp -d $NTP_SOURCE --dport 123\
 -m state --state NEW,ESTABLISHED -j ACCEPT

#This rule allows the Access Point  to get ntp settings:
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
           $IPTABLES -A forward_rules -p udp -d $NTP_SOURCE  --dport 123 -m state\
 --state NEW,ESTABLISHED -j LOG --log-level DEBUG --log-prefix "AccessPoint to Ntp. "
fi
           $IPTABLES -A forward_rules -p udp -d $NTP_SOURCE --dport 123\
 -m state --state NEW,ESTABLISHED -j ACCEPT

#This rule allows wireless devices to get ntp settings from the Win2k PDC emulator
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
           $IPTABLES -A forward_rules -p udp -d $WIN2K_NTP_SOURCE --sport 123\
 --dport 123 -m state --state NEW,ESTABLISHED -j LOG --log-level DEBUG\
 --log-prefix "fw: Wireless to Win2kNtp. "
fi
           $IPTABLES -A forward_rules -p udp -d $WIN2K_NTP_SOURCE --sport 123\
 --dport 123 -m state --state NEW,ESTABLISHED -j ACCEPT


##############################
#Section 9. Now ..... modify the rules based on the machine
#
#
#
##DHCP
#Make sure that all DHCP traffic from the selected machines can get through
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
           $IPTABLES -A output_$INT_IF -o $INT_IF -p udp -d $DHCP_SERVER_1\
  --dport 67 -m state --state NEW,ESTABLISHED -j LOG --log-level\
 DEBUG --log-prefix "out_$INT_IF to Dhcp1. "
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
fi
            $IPTABLES -A output_$INT_IF -o $INT_IF -p udp -d $DHCP_SERVER_1\
 --dport 67 -m state --state NEW,ESTABLISHED -j ACCEPT

if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A forward_rules -p udp -d $DHCP_SERVER_1  --dport 67\
 -m state --state NEW,ESTABLISHED -j LOG --log-level DEBUG --log-prefix\
 "fw: to Dhcp1. "
fi
            $IPTABLES -A forward_rules -p udp -d $DHCP_SERVER_1 --dport 67\
 -m state --state NEW,ESTABLISHED -j ACCEPT

if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A forward_rules -p udp -s $DHCP_SERVER_1 --sport 68\
 -m state --state ESTABLISHED -j LOG --log-level DEBUG --log-prefix "fw: fr Dhcp1. "
fi
            $IPTABLES -A forward_rules -p udp -s $DHCP_SERVER_1 --sport 68\
 -m state --state ESTABLISHED -j ACCEPT


if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A output_$INT_IF -o $INT_IF -p udp -d $DHCP_SERVER_2\
  --dport 67 -m state --state NEW,ESTABLISHED -j LOG --log-level\
 DEBUG --log-prefix "out_$INT_IF to Dhcp2. "
fi
            $IPTABLES -A output_$INT_IF -o $INT_IF -p udp -d $DHCP_SERVER_2\
 --dport 67 -m state --state NEW,ESTABLISHED -j ACCEPT

if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A input_$INT_IF -i $INT_IF -p udp -s $DHCP_SERVER_1\
  --sport 68 -m state --state ESTABLISHED -j LOG --log-level\
 DEBUG --log-prefix "in_$INT_IF fr Dhcp1. "
fi
            $IPTABLES -A input_$INT_IF -i $INT_IF -p udp -s $DHCP_SERVER_1\
 --sport 68 -m state --state ESTABLISHED -j ACCEPT

if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A input_$INT_IF -i $INT_IF -p udp -s $DHCP_SERVER_2\
  --sport 68 -m state --state ESTABLISHED -j LOG --log-level\
 DEBUG --log-prefix "in_$INT_IF fr Dhcp2. "
fi
            $IPTABLES -A input_$INT_IF -i $INT_IF -p udp -s $DHCP_SERVER_2\
 --sport 68 -m state --state ESTABLISHED -j ACCEPT

## SNMP
            # Allow snmp queries on udp ports 161.
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A input_$INT_IF -i $INT_IF -p udp -s $SNMP_SERVER\
  --dport 161 -m state --state NEW,ESTABLISHED -j LOG --log-level\
 DEBUG --log-prefix "in_$INT_IF fr SnmpSvr. "
fi
            $IPTABLES -A input_$INT_IF  -i $INT_IF -p udp -s $SNMP_SERVER\
 --dport 161 -m state --state  NEW,ESTABLISHED -j ACCEPT


## DNS
# Allow UDP packets to DNS servers from client.
# These rules are for wireless clients
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A forward_rules -p udp -s $WIRELESS_SUBNET -d $NAMESERVER_1\
  --dport 53 -m state --state NEW,ESTABLISHED -j LOG --log-level\
 DEBUG --log-prefix "fw: Wireless to Namesvr1 "
            $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $NAMESERVER_1\
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
  --dport 53 -m state --state NEW,ESTABLISHED -j LOG --log-level\
 DEBUG --log-prefix "fwDrop: Wireless to Namesvr1 "
fi
            $IPTABLES -A forward_rules -p udp -s $WIRELESS_SUBNET -d $NAMESERVER_1\
 --dport 53 -m state --state NEW,ESTABLISHED -j ACCEPT
            $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $NAMESERVER_1\
 --dport 53 -m state --state NEW,ESTABLISHED -j DROP

            #just to be sure there is a second nameserver listed:
            if [ $NAMESERVER_2 ]; then
                        if [ $LOG_GENERATOR = "1" ]; then #Log associated traffic
                        $IPTABLES -A forward_rules -p udp -s $WIRELESS_SUBNET\
 -d $NAMESERVER_2  --dport 53 -m state --state NEW,ESTABLISHED -j LOG\
 --log-level DEBUG --log-prefix "fw: Wireless to Namesvr2 "
                        $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET\
 -d $NAMESERVER_2  --dport 53 -m state --state NEW,ESTABLISHED -j LOG\
 --log-level DEBUG --log-prefix "fwDrop: Wireless to Namesvr2 "
                        fi
            $IPTABLES -A forward_rules -p udp -s $WIRELESS_SUBNET -d $NAMESERVER_2\
 --dport 53 -m state --state NEW,ESTABLISHED -j ACCEPT
            $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $NAMESERVER_2\
 --dport 53 -m state --state NEW,ESTABLISHED -j DROP
            fi

# These rules are for the local router
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A output_$INT_IF -p udp -o $INT_IF -d $NAMESERVER_1\
  --dport 53 -m state --state NEW,ESTABLISHED -j LOG --log-level\
 DEBUG --log-prefix "fw: router to Namesvr1 "
fi
            $IPTABLES -A output_$INT_IF -p udp -o $INT_IF -d $NAMESERVER_1\
 --dport 53 -m state --state NEW,ESTABLISHED -j ACCEPT
            #just to be sure there is a second nameserver listed:

            if [ $NAMESERVER_2 ]; then
            if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
                        $IPTABLES -A output_$INT_IF -p udp -o $INT_IF -d $NAMESERVER_2\
  --dport 53 -m state --state NEW,ESTABLISHED -j LOG --log-level\
 DEBUG --log-prefix "fw: router to Namesvr2 "
            fi
            $IPTABLES -A output_$INT_IF -p udp -o $INT_IF -d $NAMESERVER_2\
 --dport 53 -m state --state NEW,ESTABLISHED -j ACCEPT
            fi

## TELNET
# Allow telnet outbound to the vax only.
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $VAX_SERVER\
  --dport 23 -m state --state NEW,ESTABLISHED -j LOG --log-level\
 DEBUG --log-prefix "fw: Wireless to Vax "
fi
            $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $VAX_SERVER\
 --dport 23 -m state --state NEW,ESTABLISHED -j ACCEPT

## FTP
# Allow ftp outbound to an intranet server.
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $INTRANET_SERVER\
  --dport 21 -m state --state NEW,ESTABLISHED -j LOG --log-level\
 DEBUG --log-prefix "fw: Wireless to home "
fi
            $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $INTRANET_SERVER\
 --dport 21 -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
## HTTP; HTTPS
# Allow web browsing outbound.
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
           $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET\
 -m multiport --dport $WEB_BROWSER_PORTS -m state --state NEW,ESTABLISHED,RELATED\
 -j LOG --log-level DEBUG --log-prefix "fw: Wireless browsing. "
fi
           $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -m multiport\
 --dport $WEB_BROWSER_PORTS -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT

## SMTP
           # Allow all outgoing mail
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
           $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $SMTP_SERVER\
 --dport 25 -m state --state NEW,ESTABLISHED -j LOG --log-level\
 DEBUG --log-prefix "fw: Wireless to MailSvr. "
fi
           $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $SMTP_SERVER\
 --dport 25 -m state --state NEW,ESTABLISHED -j ACCEPT

if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
           $IPTABLES -A output_$INT_IF -o $INT_IF -p tcp -s $IPADDR_INT\
 -d $SMTP_SERVER --dport 25 -m state --state NEW,ESTABLISHED -j LOG\
 --log-level DEBUG --log-prefix "out_$INT_IF: rtr to MailSvr. "
fi
           $IPTABLES -A output_$INT_IF -o $INT_IF -p tcp -s $IPADDR_INT\
 -d $SMTP_SERVER --dport 25 -m state --state NEW,ESTABLISHED -j ACCEPT

## PRINTING
# Allow printing.
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
           $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET\
 -m multiport --dport $PRINTING_PORTS -m state --state NEW,ESTABLISHED,RELATED\
 -j LOG --log-level DEBUG --log-prefix "fw: Wireless to PrintPrts. "
fi
           $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET\
 -m multiport --dport $PRINTING_PORTS -m state --state NEW,ESTABLISHED,RELATED\
 -j ACCEPT


## LOGIN (Windows AD Searches)
           # Allow kerberos and ldap port access outbound to 88, 389, 1026:1029.
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
           $IPTABLES -A forward_rules -p udp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_1\
 -m multiport --dport $LOGIN_PORTS_UDP -m state --state NEW,ESTABLISHED -j LOG\
 --log-level DEBUG --log-prefix "fw: Wireless to LogSvr1,udp "
fi
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
           $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_1\
 -m multiport --dport $LOGIN_PORTS_TCP -m state --state NEW,ESTABLISHED -j LOG\
 --log-level DEBUG --log-prefix "fw: Wireless to LogSvr1,tcp "
fi
           $IPTABLES -A forward_rules -p udp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_1\
 -m multiport --dport $LOGIN_PORTS_UDP -m state --state NEW,ESTABLISHED -j ACCEPT
           $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_1\
 -m multiport --dport $LOGIN_PORTS_TCP -m state --state NEW,ESTABLISHED -j ACCEPT


if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
           $IPTABLES -A forward_rules -p udp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_2\
 -m multiport --dport $LOGIN_PORTS_UDP -m state --state NEW,ESTABLISHED -j LOG\
 --log-level DEBUG --log-prefix "fw: Wireless to LogSvr2,udp "
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
fi
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_2\
 -m multiport --dport $LOGIN_PORTS_TCP -m state --state NEW,ESTABLISHED -j LOG\
 --log-level DEBUG --log-prefix "fw: Wireless to LogSvr2,tcp "
fi
            $IPTABLES -A forward_rules -p udp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_2\
 -m multiport --dport $LOGIN_PORTS_UDP -m state --state NEW,ESTABLISHED -j ACCEPT
            $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_2\
 -m multiport --dport $LOGIN_PORTS_TCP -m state --state NEW,ESTABLISHED -j ACCEPT

if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A forward_rules -p udp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_3\
 -m multiport --dport $LOGIN_PORTS_UDP -m state --state NEW,ESTABLISHED -j LOG\
 --log-level DEBUG --log-prefix "fw: Wireless to LogSvr3,udp "
fi
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_3\
 -m multiport --dport $LOGIN_PORTS_TCP -m state --state NEW,ESTABLISHED -j LOG\
 --log-level DEBUG --log-prefix "fw: Wireless to LogSvr3,tcp "
fi
            $IPTABLES -A forward_rules -p udp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_3\
 -m multiport --dport $LOGIN_PORTS_UDP -m state --state NEW,ESTABLISHED -j ACCEPT
            $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_3\
 -m multiport --dport $LOGIN_PORTS_TCP -m state --state NEW,ESTABLISHED -j ACCEPT

if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A forward_rules -p udp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_4\
 -m multiport --dport $LOGIN_PORTS_UDP -m state --state NEW,ESTABLISHED -j LOG\
 --log-level DEBUG --log-prefix "fw: Wireless to LogSvr4,udp "
fi
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_4\
 -m multiport --dport $LOGIN_PORTS_TCP -m state --state NEW,ESTABLISHED -j LOG\
 --log-level DEBUG --log-prefix "fw: Wireless to LogSvr4,tcp "
fi
            $IPTABLES -A forward_rules -p udp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_4\
 -m multiport --dport $LOGIN_PORTS_UDP -m state --state NEW,ESTABLISHED -j ACCEPT
            $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_4\
 -m multiport --dport $LOGIN_PORTS_TCP -m state --state NEW,ESTABLISHED -j ACCEPT

if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A forward_rules -p udp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_5\
 -m multiport --dport $LOGIN_PORTS_UDP -m state --state NEW,ESTABLISHED -j LOG\
 --log-level DEBUG --log-prefix "fw: Wireless to LogSvr5,udp "
fi
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_5\
 -m multiport --dport $LOGIN_PORTS_TCP -m state --state NEW,ESTABLISHED -j LOG\
 --log-level DEBUG --log-prefix "fw: Wireless to LogSvr5,tcp "
fi
            $IPTABLES -A forward_rules -p udp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_5\
 -m multiport --dport $LOGIN_PORTS_UDP -m state --state NEW,ESTABLISHED -j ACCEPT
            $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_5\
 -m multiport --dport $LOGIN_PORTS_TCP -m state --state NEW,ESTABLISHED -j ACCEPT

if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A forward_rules -p udp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_6\
 -m multiport --dport $LOGIN_PORTS_UDP -m state --state NEW,ESTABLISHED -j LOG\
 --log-level DEBUG --log-prefix "fw: Wireless to LogSvr6,udp "
fi
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
            $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_6\
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
 -m multiport --dport $LOGIN_PORTS_TCP -m state --state NEW,ESTABLISHED -j LOG\
 --log-level DEBUG --log-prefix "fw: Wireless to LogSvr6,tcp "
fi
           $IPTABLES -A forward_rules -p udp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_6\
 -m multiport --dport $LOGIN_PORTS_UDP -m state --state NEW,ESTABLISHED -j ACCEPT
           $IPTABLES -A forward_rules -p tcp -s $WIRELESS_SUBNET -d $LOGIN_SERVER_6\
 -m multiport --dport $LOGIN_PORTS_TCP -m state --state NEW,ESTABLISHED -j ACCEPT


## SMB (Windows Protocol)
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
           $IPTABLES -A forward_rules -p tcp -m multiport --dport $SMB_PORTS_TCP\
 -m state --state NEW,ESTABLISHED -j LOG --log-level DEBUG\
 --log-prefix "fw: SmbPorts, tcp "
fi
if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
           $IPTABLES -A forward_rules -p udp -m multiport --dport $SMB_PORTS_UDP\
 -m state --state NEW,ESTABLISHED -j LOG --log-level DEBUG\
 --log-prefix "fw: SmbPorts, udp "
fi

           $IPTABLES -A forward_rules -p tcp -m multiport --dport $SMB_PORTS_TCP\
 -m state --state NEW,ESTABLISHED -j ACCEPT
           $IPTABLES -A forward_rules -p udp -m multiport --dport $SMB_PORTS_UDP\
 -m state --state NEW,ESTABLISHED -j ACCEPT

## Wireless Access (TACACS; syslog, etc.)
#We'll always log radius traffic

           $IPTABLES -A forward_rules -p tcp -m multiport --dport $RADIUS_PORTS\
 -m state --state NEW,ESTABLISHED -j LOG --log-level DEBUG\
 --log-prefix "fw: RadiusPorts, tcp "
           $IPTABLES -A forward_rules -p tcp -m multiport --dport $RADIUS_PORTS\
 -m state --state NEW,ESTABLISHED -j ACCEPT

           $IPTABLES -A forward_rules -p udp -m multiport --dport $RADIUS_PORTS\
 -m state --state NEW,ESTABLISHED -j LOG --log-level DEBUG\
 --log-prefix "fw: RadiusPorts, udp "
           $IPTABLES -A forward_rules -p udp -m multiport --dport $RADIUS_PORTS\
 -m state --state NEW,ESTABLISHED -j ACCEPT

if [ $LOG_GENERATOR = "1" ]; then #Log traffic associated with the rule
           $IPTABLES -A forward_rules -p udp -d $SYSLOG_SERVER --dport 514\
 -m state --state NEW,ESTABLISHED -j LOG --log-level DEBUG\
 --log-prefix "fw: Syslog traffic "
fi
           $IPTABLES -A forward_rules -p udp -d $SYSLOG_SERVER --dport 514\
 -m state --state NEW,ESTABLISHED -j ACCEPT

##############################
# LOGGING
#
#
# Any udp not already allowed is logged and then dropped.
#Just drop the udp traffic we don't need.  but leave these rules for debug purposes:
#$IPTABLES -A input_$EXT_IF  -i $EXT_IF -p udp -j LOG --log-level DEBUG\
# --log-prefix "in_$EXT_IF: UDP "
$IPTABLES -A input_$EXT_IF  -i $EXT_IF -p udp -j DROP

#$IPTABLES -A output_$EXT_IF -o $EXT_IF -p udp -j LOG --log-level DEBUG\
# --log-prefix "out_$EXT_IF: UDP"
$IPTABLES -A output_$EXT_IF -o $EXT_IF -p udp -j DROP
```

©SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
#Just drop the udp traffic we don't need.  but leave these rules for debug purposes:
#$IPTABLES -A input_$INT_IF  -i $INT_IF -p udp -j LOG --log-level DEBUG\
# --log-prefix "in_$INT_IF: UDP"
$IPTABLES -A input_$INT_IF  -i $INT_IF -p udp -j DROP
#$IPTABLES -A output_$INT_IF -o $INT_IF -p udp -j LOG --log-level DEBUG\
# --log-prefix "out_$INT_IF: UDP"
$IPTABLES -A output_$INT_IF -o $INT_IF -p udp -j DROP

# Any tcp not already allowed is logged and then dropped.
$IPTABLES -A input_$EXT_IF  -i $EXT_IF -p tcp -j LOG --log-level DEBUG\
 --log-tcp-options --log-prefix "in_$EXT_IF: TCP "
$IPTABLES -A input_$EXT_IF  -i $EXT_IF -p tcp -j DROP
$IPTABLES -A output_$EXT_IF -o $EXT_IF -p tcp -j LOG --log-level DEBUG\
 --log-tcp-options --log-prefix "out_$EXT_IF: TCP"
$IPTABLES -A output_$EXT_IF -o $EXT_IF -p tcp -j DROP

$IPTABLES -A input_$INT_IF  -i $INT_IF -p tcp -j LOG --log-level DEBUG\
 --log-tcp-options --log-prefix "in_$INT_IF: TCP"
$IPTABLES -A input_$INT_IF  -i $INT_IF -p tcp -j DROP
$IPTABLES -A output_$INT_IF -o $INT_IF -p tcp -j LOG --log-level DEBUG\
 --log-tcp-options --log-prefix "out_$INT_IF: TCP"
$IPTABLES -A output_$INT_IF -o $INT_IF -p tcp -j DROP

# Any mac_address rule forwarded traffic not already allowed is logged and then
dropped.
# For too much detail, uncomment the following:
$IPTABLES -A mac_address -j LOG --log-level DEBUG --log-prefix "macDrop:NoAllowRule "
$IPTABLES -A mac_address -j DROP

# Any forward_rules rule forwarded traffic not already allowed is logged and then
dropped.
# For too much detail, uncomment the following:
$IPTABLES -A forward_rules -j LOG --log-level DEBUG --log-prefix "fw: NoAllowRule "
$IPTABLES -A forward_rules -j DROP

$IPTABLES -A input_$EXT_IF  -i $EXT_IF -p tcp -j LOG --log-level DEBUG\
 --log-tcp-options --log-prefix "in_$EXT_IF: TCP "
$IPTABLES -A input_$EXT_IF  -i $EXT_IF -p tcp -j DROP
$IPTABLES -A output_$EXT_IF -o $EXT_IF -p tcp -j LOG --log-level DEBUG\
 --log-tcp-options --log-prefix "out_$EXT_IF: TCP"
$IPTABLES -A output_$EXT_IF -o $EXT_IF -p tcp -j DROP

# Anything else not already allowed is logged and then dropped.
# It will be dropped by the default policy anyway....
$IPTABLES -A input_$EXT_IF  -i $EXT_IF -j LOG --log-level DEBUG\
 --log-prefix "in_$EXT_IF: X "
$IPTABLES -A input_$EXT_IF  -i $EXT_IF -j DROP
$IPTABLES -A output_$EXT_IF -o $EXT_IF -j LOG --log-level DEBUG\
 --log-prefix "out_EXT_IF: X "
$IPTABLES -A output_$EXT_IF -o $EXT_IF -j DROP

$IPTABLES -A input_$INT_IF  -i $INT_IF -j LOG --log-level DEBUG\
 --log-prefix "in_$INT_IF: X "
$IPTABLES -A input_$INT_IF  -i $INT_IF -j DROP
$IPTABLES -A output_$INT_IF -o $INT_IF -j LOG --log-level DEBUG\
 --log-prefix "out_$INT_IF: X "
$IPTABLES -A output_$INT_IF -o $INT_IF -j DROP

#This is crucial! This is the forwarding hack. It comes at the end of the script
# to ensure that no forwarding is occurring until all rules are defined

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


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
echo " "
echo " . . . Iptables script installed successfully"
echo " "
echo " "  >> /etc/scripts/NetfilterOutput
echo " . . . Iptables script installed successfully" >> /etc/scripts/NetfilterOutput
echo " "  >> /etc/scripts/NetfilterOutput
#
#Now email the output of the script to an admin for service startup verification
#
mail -s "Netfilter output on $MACHINENAME" sysadm@company.com <
/etc/scripts/NetfilterOutput
#
exit 0
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.