Global Information Assurance Certification Paper

SANS GCFW Practical

Assessment 1

Robert Schiela
GCFW Certification, v1.6a

GIAC Enterprises is an e-business which sells fortune cookie sayings online. There are four groups of people that access the computer and network resources of GIAC Enterprises: customers, suppliers, partners, and employees.[1] As GIAC Enterprises is a budding organization, expense concerns will largely affect operational decisions. Because of this, we are planning on using various open-source software solutions. This way, our main expense will be hardware, and we will have a low software expense. One exception to this will be the database program, Oracle. However, we are choosing this database because of it's high user base, support, performance, and features.

User Description

We will assume that customers will interface with GIAC Enterprises with one of two methods: web access to purchase online fortunes and email to communicate with GIAC Enterprises when needed. Customers will connect to a production web server, with e-commerce software, to purchase online fortunes. They will be able to create and administrate their accounts on the server, and will be able to set up billing information for themselves. The entire purchasing process will be online, through this web server. For more dedicated servicing and to allow for more scaling possibilities, the web server will be driven by a database which will be installed on a second computer. To better secure the actual data, the database server will be placed on the internal network, not on the service network. GIAC Enterprises will also have an SMTP server for people on the Internet to be able to send GIAC email. This would include customers sending email to GIAC Enterprises with questions or concerns, or other customer service issues.

We will assume that suppliers will interface with GIAC Enterprises in a similar way as customers, web access and email. Suppliers will connect to a web server (different than the customer's production web server) that will allow them to submit fortunes. Basically, they will log into an account, and submit their fortunes so they are credited appropriately. Again, this will be driven by a database system on a different machine which will be connected to the internal network. In fact, it can be the same database server as the customer's, if the database server doesn't have performance issues. We will assume that the single database server can handle the load of both the customers and suppliers. Of course, the database will have to be configured to ensure that the data available to one group is not available to the other group. The suppliers will also be able to communicate to GIAC Enterprises by email through GIAC's SMTP server.

Partners will need to have more direct and secure access to GIAC Enterprises' resources. To ensure protection, an SSH server will be used as a "light VPN" solution.[2][3] Although this solution has many limitations, this solution was chosen based on cost. Partners and GIAC employees will need to share files. Partners will also need to access information on the database. Partners will share files with the SSH server. They will be able to use "scp" or "sftp" to copy files back and forth from the server. These utilities, using the SSH protocol, are encrypted and secure. This will also give the partners the option of which clients they use, either low-cost/free versions, or more expensive clients which have more features. For better security, we will place this server on a separate "partner" service network. We will also place a web server on this network. Partners will need secure/encrypted access to an interface to the database. The interface will be a web application and partners will access this web server to access appropriate information. Partners will also be able to send email to GIAC employees through the standard SMTP server. PGP and signatures will be used to encrypt and authenticate sensitive mail messages between GIAC and Partners.

GIAC Enterprises' employees will have workstations to do their work. Employees will need to communicate to customers, suppliers, and partners via email. They will also need to get information from the GIAC database. For these tasks, a web server will be installed on the internal network with appropriate interfaces driven by the database server, which, as stated above, is on the same internal network. They may also need to share documents with each other, or with partners. A private file server (just for GIAC employees) will be located on the internal network. Employees will also have access to the partner's ssh server mentioned above. This will be their solution to sharing documents with partners. It has been decided, for security reasons, that these are the only services that will be needed for general employees. Therefore no other access will be given to the employee's workstations, and there will be no direct connections between the internal network and the Internet. This policy is made to help protect the employee's workstations from unknowingly compromising their machine by malicious web or other content.

Administrators will need a way to administrate the machines on all of the networks. They will also need a way to access software updates and patches. To administrate the machines, we will allow SSH traffic from the internal network to the other networks. We will install SSH on all of the servers using tcpwrappers, and configure tcpwrappers to only allow connections from our internal network. For accessing software updates, we will allow http/https traffic from a particular machine on the service network.

Currently, it is decided that we will not need remote access for employees. All work by employees will be done from inside the internal network.

Below is the user communication flow.

- Internet users (customers) : web access (http/https) : Internet -> customer web server
- Internet users (suppliers) : web access (http/https) : Internet -> supplier web server
- Internet users (anyone) : mail access (smtp) : Internet -> mail relay server, mail relay server -> mail server
- Partners : ssh access : partner's network -> partner ssh server
- Partners : ssh access : partner's network -> partner web server
- Employees : mail access (send) (smtp) : mail server -> mail relay server, mail relay server -> Internet
- Employees : mail access (recv) (imap) : workstations -> mail server (on the same network--no firewall change needed)
- Employees : web access (http/https) : workstations -> employee web server (on the same network--no firewall change needed)
- Employees/Administrators (for partner ssh server access and administration) : ssh access : internal network -> partner vpn network, internal network -> service network
- Administrators : web access (http/https) : one particular machine on service network -> Internet

Basic Network Architecture

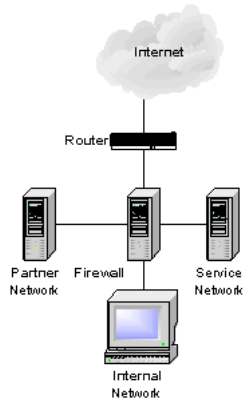
We will configure our firewall in a four-interface configuration, with a service network, a more protected partner "vpn" network, and a very protected internal network. We will have a router at our perimeter, which will connect us to our Internet Service Provider. We will configure this router to handle static packet filtering as a first layer of defense. Behind this router, we will connect our firewall. The firewall will be connected to our service network, our partner network, and our internal private network.

As stated above, we will need a web server for customers and a different web server for suppliers on our service network. These will both be Apache web servers located on the service network, with an Oracle database backend located on the internal network. Customers and suppliers will use both http and https so that all confidential transactions are secure. We will also need a mail relay server on our service network so email can be used to communicate between GIAC Enterprises and other entities on the Internet. This will be a Sendmail server. Additional utility servers will be located on the service network as well. These include the external DNS server, two ntp servers, and an

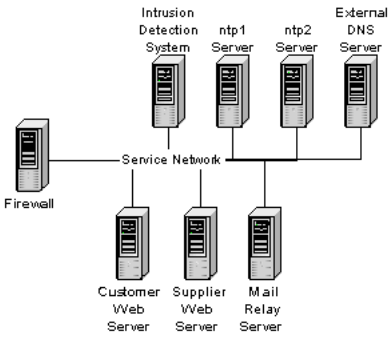
intrusion detection system.

When considering the architecture of our internal network, we want to limit the ability for intruders to affect our network, as well as protecting the employees from harming it unintentionally as well. As stated above, employees need to communicate with people on the Internet through email. However, if the employee's workstations were connected to the Internet, not only would it be possible for Internet computers to attempt to find and exploit vulnerabilities, but it would also be possible for users to unknowingly (or knowingly) open their computers up to vulnerabilities, or even infect their computer's directly. Since the only Internet network service needed for business purposes from the internal network is email, we do not need to allow network traffic from other services to come from or go to the Internet. Therefore, for security reasons, we will configure the network to not allow any of the workstations to talk to the Internet or service network directly for any services. Employees will need to be able to read and send email, but they do not need to look at Internet web sites. For email support, we will install the email server on this internal network. Workstations will connect directly to this server, both to receive and send mail. Mail sent from the workstations to this mail server will be forwarded to the mail relay server on the service network, where it will be dispatched to the Internet. Employees may need to look at appropriate GIAC data in the database as well. To support this, workstations will connect to a web server located on this internal network. Since the database is on this network as well, we will not have to make any firewall filter rules for this transaction to take place. However, even with all of these precautions, we will connect the internal network (and all of the workstations) directly to the firewall. That way, if it is decided to change this policy in the future, and allow direct connections between the workstations and the Internet, the network infrastructure will be in place. Only a policy change on the firewall will be needed to grant workstations access to the Internet. However, this means that a misconfigured firewall could make our internal network vulnerable to compromise. Again, there will be several utility servers on the internal network as well. These will include an internal DNS server, and a syslog server.

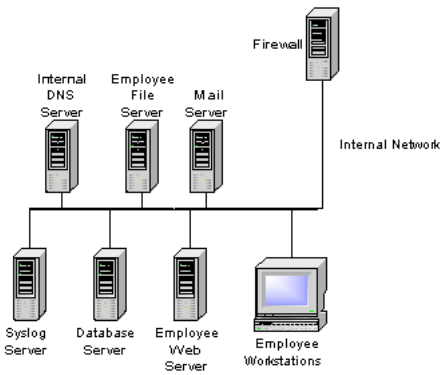
Our partner network will be fairly simple. We will install an ssh server that both the partners and the GIAC employees will have access to. We will also install a partner web server, to give partners access to information on our database. We will install several extra layers of security on these machines and this network, since it will be accessible from the Internet. First, we will configure the firewall to only allow traffic into this network that is from our internal network or one of the partners' networks. This should keep the majority of the Internet users out of this network. On the partner ssh server, we will also install ssh to use tcpwrappers, and configure ssh to only accept connections from our internal network or the partners' networks as well. Of course, all web traffic to the partner web server will use secure channel (SSL/TLS). Finally, we will install an intrusion detection system on this network for added security.



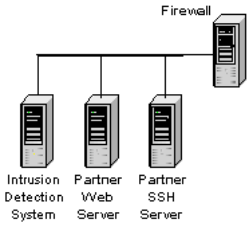
Network Architecture Diagram



Service Network Diagram



Internal Network Diagram



Partner Network Diagram

Component Details

Perimeter

The first layer of defense of our network is our router. We will be using a Cisco 3620 router, running IOS version 12.0. The purpose of the router is to connect our network to the network of our ISP, and the Internet beyond that. The router routes packets from our network to other networks, getting it one step closer to its final destination. Many routers also have the ability to do packet filtering as well: either allowing data to pass through the router, or blocking it at the source, depending on properties of the information, such as it's destination, source, or the port/service it's connecting to. We will be using static packet filtering on our router. This is the simplest kind, and is the least processor intensive. It will allow for protection, while not over-burdening the router and allowing the router to route traffic as it was intended. Since static packet filtering doesn't give us many options, we will only filter using the most simple and general rules we are concerned about. Static packet filtering is best at filtering with unconditional (or few-conditional) rules that decide to block or pass based on only one or two properties. A good example of this is blocking all traffic from a set of hosts, or blocking all traffic destined for a particular service that is not supported on our network. The router is placed at the front of our network, so all traffic coming from the Internet has to go through the router. Therefore, it's the best place to filter any traffic that we know we do not want in our network space at all. It also allows us to block traffic from leaving are network that we are sure we don't want to leave. We will configure the router to have a default allow setting. This way, any traffic that is not specifically dropped by a rule will be passed. For security, we will not allow any direct network access to the router. Any configuration changes must be made on the console connection. We will enable logging, but since the syslog server is on the internal network, and we do not want to open a path from the Internet interface of the firewall to the internal network, all logging will stay on the router.

The next layer of defense is the firewall. We will be using a Netfilter (iptables) firewall, version 1.2.6a, running on a SuSE Linux box, release 7.2 (kernel 2.4.4-4GB). A firewall has more functionality and options for its decision processing to filter traffic. We will be using stateful packet inspection, which differentiates between new connections and existing connections. A stateful packet filter keeps a state table of existing connections. When traffic comes to the packet filter, it first checks the new data against the existing connection list in the state table. If it matches, it passes the data through, else it compares the traffic to the rules. When a new connection is established, the packet filter adds this connection to the state table, so that later traffic in this connection will pass. We will also configure the firewall to have a default deny setting. Then, any traffic that is not specifically passed by a rule will be blocked. Many firewalls also perform stateful inspection. With stateful inspection, the firewall has some understanding of the higher-layer protocol being used. The firewall can make decisions based on the data payload itself, since it understands the information in the payload. Although this makes firewalls much more flexible, we will not be using stateful inspection at this time, based on time-costs to implement. We will, however, plan to add this feature, both for added security and to support non-standard protocols if needed (such as active ftp).

The firewall is connected to the service network, the partner network, and the internal network. In fact, it behaves as a router, connecting these three networks together. We will configure the firewall to protect all three networks. The firewall will be configured to allow traffic directed to valid services available on servers in the service network. However, we will configure the firewall to be much more critical of data going to our internal network. In fact, we are going to stop all traffic directly between our internal network and the Internet.

For extra security, we will also configure the firewall to implement network address translation (NAT). We will configure all of the machines on all of our networks with private IP addresses, including the internal router interface and all of the firewall interfaces. We will then use "static" NAT on the firewall to translate our public Internet addresses to the corresponding hosts. We will be using this configuration for several reasons. First, it gives us more control over our public IP addresses. We will be able to manage the assignments of our public IP addresses on the firewall. We can also separate our single public IP network into several real networks, without needing additional network space. For instance, we can assigning our partner SSH server a public IP address in the same network space as our customer web server's public IP address, even though these machines are on different real networks with different security. Using NAT also gives us a blanket of security on hosts that do not need Internet access. By not assigning public Internet addresses to hosts that do not need Internet access, we are helping to ensure that these machines cannot receive or sent Internet traffic. Also, since we are configuring our networks to have private IP addresses, we can configure all of the firewall's interfaces and the router's interface to have private addresses as well. This will help protect the router and firewall as users on the Internet will not be able to send traffic directly to them, since they do not have public addresses. Using NAT does a lot for us both in manageability of our networks and in security of our network from the Internet.

Finally, we will install an auditing program, Tripwire version 2.3-47, on all of our servers and the firewall. This program records information about many of the system files of the server and tracks any changes to any these files. It does this by making a message digest (MD5) [4] hash of the data, using one-way encryption technology. By checking this recorded information, we can tell if and when any of these files have been changed on the server. Of course, we will need to make it policy to check the signatures just before making any system changes, and to record the changes we made right after making those system changes . This way we won't get false positives caused by administrative work.

Service Network

On our service network, we will place an Intrusion Detection System (IDS). We will be using SNORT version 1.8.4 on a SuSE Linux box, release 7.2 (kernel 2.4.4-4GB). An IDS watches all network traffic and compares it to known signatures of malicious network behavior. If suspicious behavior is found, it makes a log entry and can be configured to alert as well. This is used to alert administrators that suspicious activities are occurring (or have occurred), so administrators can react to protect the network. We must note that a network IDS does not have all of the features of a host based IDS, but it is easier to manage, which is why we've chosen to use a network IDS. We are placing the IDS on the service network since this network will receive traffic from the Internet. Because the service network is "connected" to the Internet, this is the most insecure area. Adding the IDS will help us monitor the activity that occurs on this network. We will also block all Internet activity between the IDS and the Internet, as the IDS will not have a public IP address assigned to it. We will also configure the IDS to send its alerts to the syslog server. This way any IDS alerts will be placed on a different server as soon as possible, and will be centralized for easier monitoring.

Also, on our service network, we will have two ntp servers, ntp1 and ntp2, running ntp version 4.1.1. This server will be used to synchronize the time of all of our servers. This will help us when reviewing logs so we can accurately compare activity on all of our servers together. We will synchronize these ntp servers' system clocks with four stratum-2 public ntp servers, two for each server. For ntp1, we will use ntp-1.mcs.anl.gov (140.221.9.20) (Argonne National Laboratory) and ntp.cmr.gov (140.162.8.3) (Center of Seismic Studies). For ntp2, we will use and ntp-2.mcs.anl.gov (140.221.9.6) (Argonne National Laboratory) and louie.udel.edu (128.4.40.12) (University of Delaware).[5] We will synchronize our other servers off of our ntp server. Of course, we will follow ntp etiquette and notify the administrators of the stratum-2 ntp servers that we will be using them. We will be using two ntp servers for redundancy of our ntp servers themselves and of our connection to the stratum-2 ntp servers. We are placing the ntp servers on the service network since the ntp servers will need to connect to the stratum-2 servers on the Internet and we have decided not to allow connections between

our internal network and the Internet. As the ntp service is fairly low bandwidth and is not a highly critical service, these ntp servers can be very low end machines.

The ntp2 server does not have any critical information on it, and it usually has minimal load and can handle other functions. Also, its primary service is to be redundant to the ntp1 server. Because of this, we have chosen the ntp2 server to be our "gateway to the web." As we mentioned before, administrators, and possibly other employees, will need Internet web access, mainly to download files. We will use this server as that gateway. Users will ssh to this machine, and will forward X-windows through ssh. Users will then use a standard web browser that is installed on this machine to access the Internet. We will allow both http and https from this machine. This service will not be used by all employees. It will only be used when needed, mainly for administrative purposes. Access will be limited by shell accounts on the ntp2 server.

We will also have an external DNS server on the service network. This DNS server will be the authoritative DNS server for the servers on the service network. It will be the job of this server to do name resolution of our service servers for Internet users. We will not put entries of the internal network in this server; it will only know about the service network and partner network servers. It will also only know the public IP addresses of those servers and will be configured to not Also, we will configure the DNS query output to be less than 512 bytes. This way, we will be able to block DNS connection attempts through TCP, which might allow a zone transfer.[6] We will also configure this DNS server to be non-recursive. If our DNS server doesn't have an answer to a DNS query that is forwarded to it, instead of working to find the answer, it will only reply the address of its root server. We do this both for security and to limit the processing spent on each DNS query.

We will have a mail-relay server on this network. Mail coming into or going out of our environment will go through this mail relay server. This way, our main mail server, which will hold all of the email, will be protected in our internal network, and will not be directly accessible from the Internet. Also, we will be able to filter incoming viruses and spam mail at the mail-relay to help protect our users. We can also edit outgoing mail headers to hide any sensitive network environment information that may give malicious Internet users hints on how to break into our network. Obviously, this mail-relay server needs to be on the service network so it can communicate with machines on the Internet, and it is also able to communicate with our mail server, located on the internal network.

Finally, we will have our two web servers which will provide access to data to our customers and suppliers. They are separate machines to help ensure that a user in one group cannot access data that should only be for the other group. These two web servers will obviously need to be accessible from the Internet. Also, they will need to communicate with the database server which is located on the internal network. This may add network overhead, but it adds security to the database as it's not directly accessible from the Internet. Also, it allows the web server machines to focus on serving web pages, rather than processing database queries. It's also a configuration that easily allows for scalability of adding web servers connected to the single database.

Internal Network

We will place our syslog server on our internal network. This will make it less vulnerable to attack from hosts on the Internet. We have a centralized syslog server to receive the logs from all of the other servers. This makes management and monitoring of the logs easier, since they are all on one server. It also protects the logs from being modified from a malicious source. If a host is compromised, an intruder would usually try to modify the logs to hide the intrusion. However, if the logs are on a different server, it is more difficult for an intruder to hide his/her tracks. Also, having the logs on one server may show us patterns of attacks on multiple hosts that we may not have seen if we had to monitor separate logs.

We will also have an internal DNS server. This server is in place to make hostname administration of our networks easier to manage. The internal DNS will know about all of the hosts on the internal network, partner network, and service network. This way, employees will be able to access the servers by name, and administrators will only have to manage the DNS server rather than hosts files on each of the workstations. However, it will only know the private IP addresses of all of these machines. All machines on all of the networks will use this as the default DNS server. This setup will allow all of the hosts on all of the networks to find each other by hostname, but by using the private IP addresses rather than public ones. This will make our firewall management between our networks easier, as we only have to worry about our private addresses on the internal interfaces.. Since this machine is on the internal network, and cannot communicate with the Internet, we will configure this server to be non-recursive, but to know about the root DNS server. This way, hosts on the service network will be able to query this DNS server, and it will reply the root server so the hosts can then query it. This does not give us the caching functionality and performance of a recursive DNS server, but it is more secure to have this DNS server disconnected from the Internet.

As we just mentioned, we will have our mail server on this internal network. Users will access the mail server through IMAP and SMTP. Outgoing mail will be forwarded to the mail relay server. Incoming mail will come from the mail relay server to our mail server. Since the mail server is on the same network as the users' workstations, we do not have to configure the firewall to allow traffic for their access. We only have to allow traffic between the mail server and the mail-relay server on the service network.

The database will be on this internal network as well. We will have to make sure that the database can communicate with incoming connections from the customer and supplier web servers. Since we are using Oracle, this will be done through the default SQLNet port, 1521.[7] We will not be running Oracle in a multi-threaded configuration, so we do not need to worry about opening access to other ports for the SQLNet communication.

The employee web server and file server will be on this network as well. Employees will connect to these servers from their workstations. The web server will be an interface for the employees to query the database server. The file server will be for employees to store important files, or to share files with other employees. Again, as the workstations and these servers are on the same network, we will not need to make any firewall rules specifically for this traffic.

Partner Network

The Partner Network (as opposed to our partners' networks) will consist of three machines. We will use the firewall to limit access to this network's services. Only access from our internal network and from the partners' networks will be allowed. We will also limit access from this network to our other networks and the Internet. We will only allow connections for ntp to our ntp servers on the service network, syslog to our syslog server on our internal network, and the connection from the partner web server to our database. All other traffic that is initiated from this network will be dropped. This includes DNS, so the ntp, syslog, and SQL (for the database) clients will be configured using IP addresses rather than hostnames. We are not allowing DNS because we want to ensure that our partners do not have access to query our DNS server. This means more work, since each machine will have to be configured separately, but it gives us more security.

One machine, the ssh server, will be running OpenSSH version 3.1 on a SuSE Linux server and will act as a file server, to share documents between partners and employees. They will be able to use ssh to copy files (scp or sftp) to and from our server. There is a security concern since ssh access will give partner's a shell on one of our internal networks. This is why we are limiting connectivity between this machine and our other networks to only allow ntp and syslog. We will also need to be sure that user permissions are correct on this server to prevent users from being able to access data they shouldn't. We will also configure SSH to not allow X-windows forwarding or ssh-tunneling as well, so we are sure that no other services are being accessed through SSH. This is an inexpensive VPN implementation, which is why we are using it. It also gives our partners the option of connecting to us with a free client (no implementation cost to them) or to purchase a more expensive client with extra features and a GUI. For additional security, we will configure SSH to use tcpwrappers. Tcpwrappers is a program that can allow you to manage network connection restrictions for multiple services through one interface. We will configure tcpwrappers to only allow ssh access from our internal network and from the partners' networks. This is simply another layer of security to prevent unauthorized access to the server.

We are choosing ssh as our "VPN" solution for several reasons, the most important of which is cost, both to GIAC and to GIAC's partners. We are able to create a server with open source software, so the only real cost is the hardware of the machine. Also, ssh is relatively easy to configure and to manage, so there is a time-cost benefit as well. Putting the server on its own network helps isolate this machine. This way, if the ssh server is compromised, it will not give access to our internal network. Since ssh uses strong encryption (several encryption methods including 3DES or blowfish), it's also secure from detectors on the Internet. Finally, the minimum cost of installation for our partners is almost nothing. They do not need to purchase or install any VPN gateway. They only need to install an OpenSSH client on any workstations that should connect to our ssh server. There are OpenSSH or other free SSH clients for most platforms, including various Unixes, Windows, and Macintosh. The partners also have the option of purchasing SSH client software with more features, if so desired. Finally, another reason to use SSH is that it is not affected by NAT. For instance, some implementations of IPSec, specifically authenticated header (AH) are not compatible with using NAT. We will not have to worry about our NAT configuration with the

implementation we have chosen.

SSH does have its limitations, however. One is that giving an ssh account gives shell access to the machine. This means we have to make sure the machine is well hardened, and that file permissions are appropriate on the entire system. Another limitation is that ssh can forward traffic from other protocols, but forwarded (or tunneled) ports must be configured on the machine. Other VPN solutions, such as PPTP or IPSec, allow encryption and transport of any protocol, automatically. Using another VPN solution, such as IPSec, allows users to connect to multiple hosts and protocols easily and everything is encrypted between the two networks (if using a gateway VPN solution). Again, with ssh, you can only connect to the one machine, unless you configure it to forward a connection to a different machine, which could be a major security hazard. Plus, if you enable forwarding on ssh, you have no control on what protocols or software is forwarded. This is why we are choosing not to allow port forwarding. We have isolated this machine on a separate network and are limiting access to it with the firewall and tcpwrappers. We are also monitoring all traffic that goes into or out of this network with an IDS, and can even log all ssh connections with tcpwrappers if need be.

We will also have a web server on this network. Again, access to this web server will be limited to connections from our internal network and the partners' networks. This web server will be the partners' interface to information that is stored on the database. It will be configured to only allow appropriate access to data for the partners. This server will have the Apache web server installed, and will connect to the Oracle database through the standard SQLNet port, just as the customer and supplier web servers do. For security of the data, we will only support secure channel traffic to this web server (SSL/TLS) using the standard https protocol.

Finally, we will install an intrusion detection system (IDS) on this network. Again, we will use Snort, version 1.8.4. This machine will be configured very similar to the IDS system on the service network. We are installing this IDS system for additional security. Since this network is connected to the Internet, and data on this network is extremely critical, we want to carefully review any and all traffic that comes into or tries to leave this network. Again, we will configure this server to send its alerts to the syslog server. Just like the IDS on the service network, this machine will not have a public IP address assigned to it, to protect it from access from the Internet.

Network Traffic Flow

Below is the server communication flow:

- firewall -> service network ; firewall -> ntp1 server ; ntp protocol (udp 123)
- firewall -> service network ; firewall -> ntp2 server ; ntp protocol (udp 123)
- firewall -> internal network ; firewall -> syslog server ; syslog protocol (udp 514)
- Internet -> service network ; Internet hosts -> external DNS server ; dns protocol (udp 53)
- Internet -> service network ; Internet hosts -> mail-relay ; smtp protocol (tcp 25)
- Internet -> service network ; Internet hosts -> customer web server ; http/https protocol (tcp 80/443)
- Internet -> service network ; Internet hosts -> supplier web server ; http/https protocol (tcp 80/443)
- Internet -> partner network ; partners' networks -> ssh server ; ssh protocol (tcp 22)
- Internet -> partner network ; partners' networks -> partner web server ; https protocol (tcp 443)
- service network -> Internet ; ntp1 server -> stratum2 ntp servers ; ntp protocol (udp 123)
- service network -> Internet ; ntp2 server -> stratum2 ntp servers ; ntp protocol (udp 123)
- service network -> Internet ; ntp2 server -> Internet hosts ; http/https protocol (tcp 80/443)
- service network -> Internet ; mail-relay -> Internet hosts ; smtp protocol (tcp 25)
- service network -> internal network ; mail-relay -> mail server ; smtp protocol (tcp 25)
- service network -> internal network ; customer web server -> database ; SQLNet (tcp 1521)
- service network -> internal network ; supplier web server -> database ; SQLNet (tcp 1521)
- service network -> internal network ; service hosts -> syslog server ; syslog protocol (udp 514)
- service network -> internal network ; service hosts -> internal DNS server ; dns protocol (udp 53)
- service network -> Internet ; service hosts -> Internet hosts ; dns protocol (udp/tcp 53)
- internal network -> firewall ; internal hosts -> firewall ; ssh protocol (tcp 22)
- internal network -> service network ; internal hosts -> service hosts ; ssh protocol (tcp 22)
- internal network -> partner network ; internal hosts -> partner hosts ; ssh protocol (tcp 22)
- internal network -> service network ; mail server -> mail-relay ; smtp protocol (tcp 25)
- internal network -> service network ; internal hosts -> ntp1 server ; ntp protocol (udp 123)
- internal network -> service network ; internal hosts -> ntp2 server ; ntp protocol (udp 123)
- partner network -> service network ; partner hosts -> ntp1 server ; ntp protocol (udp 123)
- partner network -> service network ; partner hosts -> ntp2 server ; ntp protocol (udp 123)
- partner network -> internal network ; partner hosts -> syslog server ; syslog protocol (udp 514)
- partner network -> internal network ; partner web server -> database ; SQLNet (tcp 1521)

Assessment 2

Security Policy

We will focus on the security policy of the router, the firewall and the VPN (SSH). We will then discuss the security policies of the other security devices we have implemented.

Router

As we mentioned, we have a Cisco router as our border router. We will use the router's static filtering ability for our first layer of security defense. We will use the router to deny specific traffic that we know we do not want into our network. We will also use it to deny traffic that we know we do not want out of our network. This will include traffic that has source hosts of private or unused IP addresses, or traffic that is using source-routing options, among others. Since we will be denying traffic coming from IP addresses that are currently IANA-reserved, we will need to regularly check the IANA IP address space assignments and update our rules for any newly assigned address spaces. For incoming traffic (from the Internet) we will set "permit any" as the default action. For outgoing (to the Internet) we will set "deny any" as the default action. For this example, we will assume that the address space 1.0.0.0/8 was recently assigned, and a portion was assigned to our ISP. We were assigned the space 1.129.1.0/24. We will configure the router's routing table as well. It will know about it's two interfaces, it will route all 1.129.1.0/24 traffic to the firewall's external interface (192.168.0.2), and the default route will be the gateway of the ISP. We will assign the router's internal Ethernet port address to be the private address of 192.168.0.1/24.

```
service password encryption
no service tcp-small-servers
no service udp-small-servers
no service finger
no ip http
no ip bootp
no cdp
no snmp
```

```
no ip source-route
no ip unreachables
no ip direct-broadcast

ip access-list extended filterin

     deny ip 10.0.0.0 0.255.255.255 any log
     deny ip 172.16.0.0 0.15.255.255 any log
     deny ip 192.168.0.0 0.0.255.255 any log
     deny ip 127.0.0.0 0.255.255.255 any log
     deny ip 1.129.1.0 0.0.0.255 any log
     deny ip 0.0.0.0 0.255.255.255 any log
     deny ip 2.0.0.0 0.255.255.255 any log
     deny ip 5.0.0.0 0.255.255.255 any log
     deny ip 7.0.0.0 0.255.255.255 any log
     deny ip 23.0.0.0 0.255.255.255 any log
     deny ip 27.0.0.0 0.255.255.255 any log
     deny ip 31.0.0.0 0.255.255.255 any log
     deny ip 36.0.0.0 0.255.255.255 any log
     deny ip 37.0.0.0 0.255.255.255 any log
     deny ip 39.0.0.0 0.255.255.255 any log
     deny ip 41.0.0.0 0.255.255.255 any log
     deny ip 42.0.0.0 0.255.255.255 any log
     deny ip 58.0.0.0 0.255.255.255 any log
     deny ip 59.0.0.0 0.255.255.255 any log
     deny ip 60.0.0.0 0.255.255.255 any log
     deny ip 69.0.0.0 3.255.255.255 any log
     deny ip 72.0.0.0 7.255.255.255 any log
     deny ip 70.0.0.0 0.255.255.255 any log
     deny ip 70.0.0.0 0.255.255.255 any log
     deny ip 82.0.0.0 0.255.255.255 any log
     deny ip 83.0.0.0 0.255.255.255 any log
     deny ip 84.0.0.0 3.255.255.255 any log
     deny ip 88.0.0.0 7.255.255.255 any log
     deny ip 96.0.0.0 31.255.255.255 any log
     deny ip 197.0.0.0 0.255.255.255 any log
     deny ip 221.0.0.0 0.255.255.255 any log
     deny ip 222.0.0.0 0.255.255.255 any log
     deny ip 223.0.0.0 0.255.255.255 any log
     deny ip 224.0.0.0 31.255.255.255 any log
     deny ip 1.166.166.166 255.255.255.255 any log
     permit any any


ip access-list extended filterout

     permit 1.129.1.0 0.0.0.255
     deny any any log
```

First, we configure the router to restrict the services offered by the router. We also configure the router to restrict some of the more risky types of traffic, such as source-routed traffic and returning ICMP unreachable messages.[8]

We then create and define our ingress filter. First we restrict any traffic that has a source IP address of reserved address for private networks.[9] Network traffic with these source addresses should never reach the Internet. Then we restrict any traffic that has a source IP address of the loopback address (127.*.*.*). Network traffic with any of these addresses should never be put on the wire (layer 1) of any network. Then we deny any traffic that has a source IP address of one of our machines. We should never see any traffic from one of our IP addresses that is coming into our router from the Internet interface. We block all of the address spaces that are currently unassigned or reserved by the IANA.[10] Since some of these addresses could be later be assigned, we will have to regularly check the official assignment document for changes. Also, please note that we have not blocked the 1.*.*.* addresses. For the scope of this paper, we have assumed that this address space was recently assigned, and we received a class C network in this space. Finally, we block any IP addresses (1.166.166.166) that have performed attacks on our network in the past. This way their traffic never even reaches our firewall and is stopped at our perimeter. After blocking all of this traffic, we are assigning a default permit rule. Any traffic that is not specifically blocked by a previous rule will be permitted to pass through the router, and into our network. Next stop for the traffic is the firewall. Note that we are logging any traffic that is blocked, and not logging that is allowed. This way, we can monitor any traffic that should not be getting to our router, but we won't be overwhelmed by logs of all the allowed traffic that we get.

For egress filtering, we are only allowing traffic that has a source IP of our public Internet address space. We then have a default deny rule. Any traffic that does not have a source of our public IP address space will be dropped. This way we are doing our duty to prevent traffic coming from one of our private IP addresses, in case our NAT is misconfigured in our firewall. As we mentioned traffic with a source address from a private network should never be routed to the Internet. So, this rule Also, this will prevent any spoofed traffic from our network, which will prevent any spoof attacks originating from our network, in case one of our servers is compromised. Note here that we again are not logging the traffic that we permit, and only log what we drop. Again, this is so we can monitor any suspicious network activity without being overwhelmed by logs of "normal" traffic.

Firewall

Our firewall will be our main layer of defense from attack. We will be using stateful packet filtering to provide protection of our network. For extra layered protection, we will copy all of our router rules to the firewall. This way, if any traffic manages to sneak past the router, the firewall will stop it. Also, we will assign a private IP address to all of interfaces of the firewall, including the external interface, so it is not directly connectable from hosts on the Internet. We will assign the external interface an address of 192.168.0.2/24, and will set the default route of the firewall to be 192.168.0.1, the router's internal interface. We will configure the firewall to have a default "deny all" action; if traffic does not match the state table or a specific rule allowing it then the traffic is dropped. We will only allow traffic that we have explicitly stated in our security policy.

We configure the firewall's other interfaces to have IP addresses of 192.168.1.1/24, 192.168.2.1/24, and 192.168.3.1/24 for the service network, partner network, and internal network, respectively. This will give us the ability to have up to 253 hosts on each network, which should be more than enough. Currently, iptables only has a command line interface. There are third-party distributions of iptables that have other interfaces, but we are using the raw iptables with no additional interface. To configure our firewall, we are going to create a startup script that is executed each time the firewall machine boots. We will create this script in the /etc/rc.d directory, and call it "firewall". We then create the appropriate links in the startup rc2.d, rc3.d, and rc5.d directories so the script is executed when the machine boots. We create symbolic links to the real file with the following commands.

```
/bin/ln -s /etc/rc.d/firewall /etc/rc.d/rc2.d/S08firewall
/bin/ln -s /etc/rc.d/firewall /etc/rc.d/rc3.d/S08firewall
/bin/ln -s /etc/rc.d/firewall /etc/rc.d/rc5.d/S08firewall
```

This will start the firewall script immediately after the network is started and the routing and interface is initialized.

Iptables stores it's configuration information and rules in several tables, one for all of the filtering rules, one for all of the NAT rules, and one for all of the "mangle" rules. Our configuration will not be using "mangle" rules, but this is usually used for marking certain packets to be used later and for quality of service functionality. Each table consists of several chains; each chain has a specific function of the type of traffic it should handle. For instance, the filtering table has three chains: one for input traffic, one for output traffic, and one for forwarded traffic. Input traffic is traffic whose final destination is the firewall. Output traffic is traffic whose original source is the firewall. Forwarded traffic is traffic that is routed through the firewall. The NAT table has three chains as well: prerouting, postrouting, and output. The prerouting chain contains NAT rules that should be translated as soon as the firewall receives the traffic. The postrouting chain contains NAT rules that should be translated just before the firewall sends the traffic. The output chain is again for traffic that was generated at the firewall, rather than being forwarded. In iptables, we use the prerouting chain for "destination NAT", or DNAT. We use postrouting chain for "source NAT" or SNAT. Configuring DNAT allows the Internet to connect to our hosts. Configuring SNAT allows our hosts to connect to the Internet. We will not be using the output chain of the NAT table in our configuration.

When network traffic arrives at the firewall, it decides which chain it should use. It then compares the traffic to the rules in the chain, one by one, in order. When it makes its first match, it performs the action described by the jump option and the processing is complete. Basically, when traffic arrives at the firewall, it first runs through the prerouting chain, then it decides the appropriate route, then it runs through the appropriate filtering chain, and finally it runs through the postrouting chain.

Iptables has a fairly simple syntax. It basically consists of the command name followed by the appropriate table, followed by the action you want to perform, followed by the appropriate chain, followed by any options. For instance:

iptables -t nat -L PREROUTING

This command uses the NAT table (-t nat) and lists (-L) the prerouting chain (PREROUTING). There is an implicit default of the filtering table (-t filter) if no other table is explicitly listed. For instance:

iptables -A FORWARD -s 10.0.0.0/8 -j DROP

This command uses the filter tables (implicit, since no other table is specified) and adds a rule (-A) to the forward chain (FORWARD). The rule is checking for a source IP address of 10.*.*.*, as designated by the mask. If any traffic were to match on this rule, it would be dropped, as specified by the jump clause (-j DROP).

With iptables, using the "add" function (-A) adds a rule at the end of the list. So, rules will be tested in the order they were "added". We will use this behavior in our script by clearing any existing rules, and then simply writing "iptables -A" commands in the order we want the rules to be processed. We will also configure the default policy of each chain. This tells the firewall how to handle any traffic that does not match any of the rules. Finally, since we are configuring iptables with a script, we will use some of the additional features that scripts give us, such as assigning and reusing variables. This makes modifications to the script easier and it makes the script easier to read.

```
#! /bin/sh
#
# Firewall rules set
#
# Author:  Robert Schiela
#
# Reference: Linux Firewalls, Second Edition; Robert L. Ziegler
#
# constants
UPLINK_IF="eth0"                # Uplink connected interface
UPLINK_NET="192.168.0.0/24"     # Uplink network IP range
UPLINK_IP="192.168.0.2"         # Uplink interface IP address
ROUTER_IP="192.168.0.1"         # private IP address for border router
SERVICE_IF="eth1"               # Service network connected interface
SERVICE_NET="192.168.1.0/24"    # Service network IP range
SERVICE_IP="192.168.1.1"        # Service network interface IP address
PARTNER_IF="eth2"               # Partner/VPN network connected interface
PARTNER_NET="192.168.2.0/24"    # Partner network IP range
PARTNER_IP="192.168.2.1"        # Partner network interface IP address
INTERNAL_IF="eth3"              # Internal connected interface
INTERNAL_NET="192.168.3.0/24"   # Internal network IP range
INTERNAL_IP="192.168.3.1"       # Internal network interface IP address
LOOPBACK_IF="lo"                # Loopback interface
INTERNET_NET="1.129.1.0/24"     # Registered Internet Address space
BROADCAST_SRC="0.0.0.0"         # broadcast source address
BROADCAST_DEST="255.255.255.255"     #broadcast destination address
PRIVPORTS="0:1023"              # well-known service port range
UNPRIVPORTS="1024:65535"        # ephemeral port range
CWEB_PUB="1.129.1.2"            # public Internet add. for cust web server
CWEB_PRIV="192.168.1.2"         # private IP address for cust web server
SWEB_PUB="1.129.1.3"            # public Internet add. for supplier web server
SWEB_PRIV="192.168.1.3"         # private IP address for supplier web server
EDNS_PUB="1.129.1.4"            # public Internet add. for external DNS server
EDNS_PRIV="192.168.1.4"         # private IP address for external DNS server
MREL_PUB="1.129.1.5"            # public Internet add. for mail relay
MREL_PRIV="192.168.1.5"         # private IP address for mail relay
NTP1_PUB="1.129.1.6"            # public Internet add. for ntp1 server
NTP1_PRIV="192.168.1.6"         # private IP address for ntp1 server
NTPS2_11="140.221.9.20"         # IP address of Stratum2 ntp server #1 for ntp1
NTPS2_12="140.162.8.3"          # IP address of Stratum2 ntp server #2 for ntp1
NTP2_PUB="1.129.1.7"            # public Internet add. for ntp2 server
NTP2_PRIV="192.168.1.7"         # private IP address for ntp2 server
NTPS2_21="140.221.9.6"          # IP address of Stratum2 ntp server #1 for ntp2
NTPS2_22="128.4.40.12"          # IP address of Stratum2 ntp server #2 for ntp2
PARTSSH_PUB="1.129.1.8"         # public Internet address for Partner SSH server
PARTSSH_PRIV="192.168.2.8"      # private IP address for Partner SSH server
PWEB_PUB="1.129.1.9"            # public Internet address for Partner Web server
PWEB_PRIV="192.168.2.9"         # private IP address for Partner Web server
SYSLOG="192.168.3.2"            # private IP address for syslog server
MSERV="192.168.3.3"             # private IP address for mail server
DBASE="192.168.3.4"             # private IP address for Database server
IDNS="192.168.3.5"              # private IP address for internal DNS server
PARTNER1_NET="1.130.0.0/24"     # Network address of Partner 1's Network
PARTNER2_NET="1.140.1.0/24"     # Network address of Partner 2's Network
PARTNER3_NET="1.150.2.0/24"     # Network address of Partner 3's Network
BADGUY1_IP="1.166.166.166"      # IP address of bad guy who attacked us before
```

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

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

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

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

# Drop spoofed packets which cannot be replied to on the received
# interface, based on routing tables
for f in /proc/sys/net/ipv4/conf/*/rp_filter; do
  echo 1 > $f
done

# Initialize chains -- clear any existing rules
iptables --flush
iptables -t nat --flush
iptables -t mangle --flush

# Enable traffic on the loopback interface
iptables -A INPUT  -i $LOOPBACK_IF -j ACCEPT
iptables -A OUTPUT -o $LOOPBACK_IF -j ACCEPT

# Set the default policy to drop, on all tables/chains
iptables --policy INPUT     DROP
iptables --policy OUTPUT    DROP
iptables --policy FORWARD   DROP

iptables -t nat --policy PREROUTING     ACCEPT
iptables -t nat --policy OUTPUT         ACCEPT
iptables -t nat --policy POSTROUTING    ACCEPT

iptables -t mangle --policy PREROUTING  ACCEPT
iptables -t mangle --policy OUTPUT      ACCEPT


# Drop packets that have illegal tcp flag settings
#   All flag bits cleared
iptables -A INPUT -p tcp --tcp-flags ALL NONE -j LOG \
      --log-prefix "SCAN? BadTCPFlags: NONE"
iptables -A INPUT -p tcp --tcp-flags ALL NONE -j DROP
iptables -A FORWARD -p tcp --tcp-flags ALL NONE -j LOG \
      --log-prefix "SCAN? BadTCPFlags: NONE"
iptables -A FORWARD -p tcp --tcp-flags ALL NONE -j DROP

#   SYN and FIN both set
iptables -A INPUT -p tcp --tcp-flags SYN,FIN SYN,FIN -j LOG \
      --log-prefix "SCAN? BadTCPFlags: SYN,FIN"
iptables -A INPUT -p tcp --tcp-flags SYN,FIN SYN,FIN -j DROP
iptables -A FORWARD -p tcp --tcp-flags SYN,FIN SYN,FIN -j LOG \
      --log-prefix "SCAN? BadTCPFlags: SYN,FIN"
iptables -A FORWARD -p tcp --tcp-flags SYN,FIN SYN,FIN -j DROP

#   SYN and RST both set
iptables -A INPUT -p tcp --tcp-flags SYN,RST SYN,RST -j LOG \
      --log-prefix "SCAN? BadTCPFlags: SYN,RST"
iptables -A INPUT -p tcp --tcp-flags SYN,RST SYN,RST -j DROP
iptables -A FORWARD -p tcp --tcp-flags SYN,RST SYN,RST -j LOG \
      --log-prefix "SCAN? BadTCPFlags: SYN,RST"
iptables -A FORWARD -p tcp --tcp-flags SYN,RST SYN,RST -j DROP

#   FIN and RST both set
iptables -A INPUT -p tcp --tcp-flags FIN,RST FIN,RST -j LOG \
      --log-prefix "SCAN? BadTCPFlags: FIN,RST"
iptables -A INPUT -p tcp --tcp-flags FIN,RST FIN,RST -j DROP
iptables -A FORWARD -p tcp --tcp-flags FIN,RST FIN,RST -j LOG \
      --log-prefix "SCAN? BadTCPFlags: FIN,RST"
iptables -A FORWARD -p tcp --tcp-flags FIN,RST FIN,RST -j DROP

#   FIN is only bit set, with no ACK
iptables -A INPUT -p tcp --tcp-flags ACK,FIN FIN -j LOG \
      --log-prefix "SCAN? BadTCPFlags: FIN, !ACK"
iptables -A INPUT -p tcp --tcp-flags ACK,FIN FIN -j DROP
iptables -A FORWARD -p tcp --tcp-flags ACK,FIN FIN -j LOG \
      --log-prefix "SCAN? BadTCPFlags: FIN, !ACK"
iptables -A FORWARD -p tcp --tcp-flags ACK,FIN FIN -j DROP

#   PSH is only bit set, with no ACK
iptables -A INPUT -p tcp --tcp-flags ACK,PSH PSH -j LOG \
      --log-prefix "SCAN? BadTCPFlags: PSH, !ACK"
iptables -A INPUT -p tcp --tcp-flags ACK,PSH PSH -j DROP
iptables -A FORWARD -p tcp --tcp-flags ACK,PSH PSH -j LOG \
      --log-prefix "SCAN? BadTCPFlags: PSH, !ACK"
iptables -A FORWARD -p tcp --tcp-flags ACK,PSH PSH -j DROP

#   URG is only bit set, with no ACK
iptables -A INPUT -p tcp --tcp-flags ACK,URG URG -j LOG \
```

```
      --log-prefix "SCAN? BadTCPFlags: URG, !ACK"
iptables -A INPUT -p tcp --tcp-flags ACK,URG URG -j DROP
iptables -A FORWARD -p tcp --tcp-flags ACK,URG URG -j LOG \
      --log-prefix "SCAN? BadTCPFlags: URG, !ACK"
iptables -A FORWARD -p tcp --tcp-flags ACK,URG URG -j DROP


# Accept connections existing in state table, in,out,forward
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT


# Set up NATing for Internet public machines
iptables -t nat -A PREROUTING -i $UPLINK_IF -d $CWEB_PUB -j DNAT \
      --to-destination $CWEB_PRIV
iptables -t nat -A PREROUTING -i $UPLINK_IF -d $SWEB_PUB -j DNAT \
      --to-destination $SWEB_PRIV
iptables -t nat -A PREROUTING -i $UPLINK_IF -d $EDNS_PUB -j DNAT \
      --to-destination $EDNS_PRIV
iptables -t nat -A PREROUTING -i $UPLINK_IF -d $MREL_PUB -j DNAT \
      --to-destination $MREL_PRIV
iptables -t nat -A PREROUTING -i $UPLINK_IF -d $NTP1_PUB -j DNAT \
      --to-destination $NTP1_PRIV
iptables -t nat -A PREROUTING -i $UPLINK_IF -d $NTP2_PUB -j DNAT \
      --to-destination $NTP2_PRIV
iptables -t nat -A PREROUTING -i $UPLINK_IF -d $PARTSSH_PUB -j DNAT \
      --to-destination $PARTSSH_PRIV
iptables -t nat -A PREROUTING -i $UPLINK_IF -d $PWEB_PUB -j DNAT \
      --to-destination $PWEB_PRIV

iptables -t nat -A POSTROUTING -o $UPLINK_IF -s $CWEB_PRIV -j SNAT \
      --to-source $CWEB_PUB
iptables -t nat -A POSTROUTING -o $UPLINK_IF -s $SWEB_PRIV -j SNAT \
      --to-source $SWEB_PUB
iptables -t nat -A POSTROUTING -o $UPLINK_IF -s $EDNS_PRIV -j SNAT \
      --to-source $EDNS_PUB
iptables -t nat -A POSTROUTING -o $UPLINK_IF -s $MREL_PRIV -j SNAT \
      --to-source $MREL_PUB
iptables -t nat -A POSTROUTING -o $UPLINK_IF -s $NTP1_PRIV -j SNAT \
      --to-source $NTP1_PUB
iptables -t nat -A POSTROUTING -o $UPLINK_IF -s $NTP2_PRIV -j SNAT \
      --to-source $NTP2_PUB
iptables -t nat -A POSTROUTING -o $UPLINK_IF -s $PARTSSH_PRIV -j SNAT \
      --to-source $PARTSSH_PUB
iptables -t nat -A POSTROUTING -o $UPLINK_IF -s $PWEB_PRIV -j SNAT \
      --to-source $PWEB_PUB

# Log and Drop traffic not allowed by router (layer of protection)
iptables -A FORWARD -s 10.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 10.0.0.0/8 -j DROP
iptables -A FORWARD -s 172.16.0.0/12 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 172.16.0.0/12 -j DROP
iptables -A FORWARD -i $UPLINK_IF -s 192.168.0.0/16 -j LOG \
      --log-prefix "Spoofed Packet passed router!"
iptables -A FORWARD -i $UPLINK_IF -s 192.168.0.0/16 -j DROP
iptables -A FORWARD -s 192.168.4.0/22 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 192.168.4.0/22 -j DROP
iptables -A FORWARD -s 192.168.8.0/21 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 192.168.8.0/21 -j DROP
iptables -A FORWARD -s 192.168.16.0/20 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 192.168.16.0/20 -j DROP
iptables -A FORWARD -s 192.168.32.0/19 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 192.168.32.0/19 -j DROP
iptables -A FORWARD -s 192.168.64.0/18 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 192.168.64.0/18 -j DROP
iptables -A FORWARD -s 192.168.128.0/17 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 192.168.128.0/17 -j DROP
iptables -A FORWARD -s 127.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 127.0.0.0/8 -j DROP
iptables -A FORWARD -s 0.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 0.0.0.0/8 -j DROP
iptables -A FORWARD -s 2.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 2.0.0.0/8 -j DROP
iptables -A FORWARD -s 5.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 5.0.0.0/8 -j DROP
iptables -A FORWARD -s 7.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 7.0.0.0/8 -j DROP
iptables -A FORWARD -s 23.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 23.0.0.0/8 -j DROP
iptables -A FORWARD -s 27.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 27.0.0.0/8 -j DROP
```

```
iptables -A FORWARD -s 31.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 31.0.0.0/8 -j DROP
iptables -A FORWARD -s 36.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 36.0.0.0/8 -j DROP
iptables -A FORWARD -s 37.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 37.0.0.0/8 -j DROP
iptables -A FORWARD -s 39.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 39.0.0.0/8 -j DROP
iptables -A FORWARD -s 41.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 41.0.0.0/8 -j DROP
iptables -A FORWARD -s 42.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 42.0.0.0/8 -j DROP
iptables -A FORWARD -s 58.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 58.0.0.0/8 -j DROP
iptables -A FORWARD -s 59.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 59.0.0.0/8 -j DROP
iptables -A FORWARD -s 60.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 60.0.0.0/8 -j DROP
iptables -A FORWARD -s 69.0.0.0/6 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 69.0.0.0/6 -j DROP
iptables -A FORWARD -s 72.0.0.0/5 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 72.0.0.0/5 -j DROP
iptables -A FORWARD -s 82.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 82.0.0.0/8 -j DROP
iptables -A FORWARD -s 83.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 83.0.0.0/8 -j DROP
iptables -A FORWARD -s 84.0.0.0/6 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 84.0.0.0/6 -j DROP
iptables -A FORWARD -s 88.0.0.0/5 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 88.0.0.0/5 -j DROP
iptables -A FORWARD -s 96.0.0.0/3 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 96.0.0.0/3 -j DROP
iptables -A FORWARD -s 197.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 197.0.0.0/8 -j DROP
iptables -A FORWARD -s 221.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 221.0.0.0/8 -j DROP
iptables -A FORWARD -s 222.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 222.0.0.0/8 -j DROP
iptables -A FORWARD -s 223.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 223.0.0.0/8 -j DROP
iptables -A FORWARD -s 224.0.0.0/3 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 224.0.0.0/3 -j DROP

# Drop IP addresses that have attacked us before
iptables -A FORWARD -s $BADGUY1_IP -j LOG \
--log-prefix "BAD GUY IP"



# Drop broadcasting traffic
#   Traffic with source of Broadcast Destination
iptables -A FORWARD -s 255.255.255.255 -j LOG \
      --log-prefix "Spoofed Packet: broadcast"
iptables -A FORWARD -s 255.255.255.255 -j DROP

#   Traffic with destination of Broadcast Source
iptables -A FORWARD -d 0.0.0.0 -j LOG \
      --log-prefix "Spoofed Packet: broadcast"
iptables -A FORWARD -d 0.0.0.0 -j DROP

#   Traffic to network address (.0) or broadcast (.255)
iptables -A FORWARD -d 192.168.0.0 -j LOG \
      --log-prefix "Packet to Network address"
iptables -A FORWARD -d 192.168.0.0 -j DROP
iptables -A FORWARD -d 192.168.1.0 -j LOG \
      --log-prefix "Packet to Network address"
iptables -A FORWARD -d 192.168.1.0 -j DROP
iptables -A FORWARD -d 192.168.2.0 -j LOG \
      --log-prefix "Packet to Network address"
iptables -A FORWARD -d 192.168.2.0 -j DROP
iptables -A FORWARD -d 192.168.0.255 -j LOG \
      --log-prefix "Packet to Broadcast address"
iptables -A FORWARD -d 192.168.0.255 -j DROP
iptables -A FORWARD -d 192.168.1.255 -j LOG \
      --log-prefix "Packet to Broadcast address"
iptables -A FORWARD -d 192.168.1.255 -j DROP
```

```
iptables -A FORWARD -d 192.168.2.255 -j LOG \
      --log-prefix "Packet to Broadcast address"
iptables -A FORWARD -d 192.168.2.255 -j DROP


# Accept incoming traffic to customer/supplier web server
iptables -A FORWARD -i $UPLINK_IF -p tcp --sport $UNPRIVPORTS -d $CWEB_PRIV \
      --dport 80 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $UPLINK_IF -p tcp --sport $UNPRIVPORTS -d $CWEB_PRIV \
      --dport 443 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $UPLINK_IF -p tcp --sport $UNPRIVPORTS -d $SWEB_PRIV \
      --dport 80 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $UPLINK_IF -p tcp --sport $UNPRIVPORTS -d $SWEB_PRIV \
      --dport 443 -m state --state NEW -j ACCEPT


# Accept incoming traffic to partner web server from partners' networks
iptables -A FORWARD -i $UPLINK_IF -p tcp -s $PARTNER1_NET --sport $UNPRIVPORTS \
      -d $PWEB_PRIV --dport 443 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $UPLINK_IF -p tcp -s $PARTNER2_NET --sport $UNPRIVPORTS \
      -d $PWEB_PRIV --dport 443 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $UPLINK_IF -p tcp -s $PARTNER3_NET --sport $UNPRIVPORTS \
      -d $PWEB_PRIV --dport 443 -m state --state NEW -j ACCEPT

# Accept communication between cweb,sweb and dbase, on standard
#      SQLNet port (1521)
iptables -A FORWARD -i $SERVICE_IF -p tcp -s $CWEB_PRIV -d $DBASE \
      --dport 1521 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $SERVICE_IF -p tcp -s $SWEB_PRIV -d $DBASE \
      --dport 1521 -m state --state NEW -j ACCEPT


# Accept communication between Partner Web server and dbase,
#     on standard SQLNet port (1521)
iptables -A FORWARD -i $PARTNER_IF -p tcp -s $PARTSSH_PRIV -d $DBASE \
      --dport 1521 -m state --state NEW -j ACCEPT


# Accept dns, udp in to dns server, udp/tcp out from all servers,
#      udp to internal dns server from service_net
iptables -A FORWARD -i $UPLINK_IF -p udp -d $EDNS_PRIV --dport 53 \
      -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $SERVICE_IF -o $UPLINK_IF -p tcp -s $SERVICE_NET \
      --dport 53 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $SERVICE_IF -o $UPLINK_IF -p udp -s $SERVICE_NET \
      --dport 53 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $SERVICE_IF -p udp -s $SERVICE_NET -d $IDNS \
      --dport 53 -m state --state NEW -j ACCEPT


# Allow web traffic from ntp2 server.  Use ntp2 server as platform
#    for downloading upgrades/updates of software.
iptables -A FORWARD -i $SERVICE_IF -o $UPLINK_IF -p tcp -s $NTP2_PRIV --dport 80 \
      -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $SERVICE_IF -o $UPLINK_IF -p tcp -s $NTP2_PRIV --dport 443 \
      -m state --state NEW -j ACCEPT


# Accept ssh traffic from Partner Networks to our Partner SSH machine
iptables -A FORWARD -i $UPLINK_IF -p tcp -s $PARTNER1_NET -d $PARTSSH_PRIV \
      --dport 22 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $UPLINK_IF -p tcp -s $PARTNER2_NET -d $PARTSSH_PRIV \
      --dport 22 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $UPLINK_IF -p tcp -s $PARTNER3_NET -d $PARTSSH_PRIV \
      --dport 22 -m state --state NEW -j ACCEPT



# Accept ntp traffic between our ntp servers,
#    and the stratum servers they access
iptables -A FORWARD -i $SERVICE_IF -p udp -s $NTP1_PRIV -d $NTPS2_11 \
      --dport 123 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $SERVICE_IF -p udp -s $NTP1_PRIV -d $NTPS2_12 \
      --dport 123 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $SERVICE_IF -p udp -s $NTP2_PRIV -d $NTPS2_21 \
      --dport 123 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $SERVICE_IF -p udp -s $NTP2_PRIV -d $NTPS2_22 \
      --dport 123 -m state --state NEW -j ACCEPT

# Accept ntp traffic between all of our private networks,
#    and firewall and our ntp servers
iptables -A FORWARD -i $INTERNAL_IF -p udp -s $INTERNAL_NET -d $NTP1_PRIV \
      --dport 123 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $INTERNAL_IF -p udp -s $INTERNAL_NET -d $NTP2_PRIV \
      --dport 123 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $PARTNER_IF -p udp -s $PARTNER_NET -d $NTP1_PRIV \
      --dport 123 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $PARTNER_IF -p udp -s $PARTNER_NET -d $NTP2_PRIV \
      --dport 123 -m state --state NEW -j ACCEPT

# Accept ntp traffic from firewall to ntp servers
iptables -A OUTPUT -o $SERVICE_IF -p udp -s $SERVICE_IP -d $NTP1_PRIV \
      --dport 123 -m state --state NEW -j ACCEPT
iptables -A OUTPUT -o $SERVICE_IF -p udp -s $SERVICE_IP -d $NTP2_PRIV \
      --dport 123 -m state --state NEW -j ACCEPT
```

```
# Accept syslog traffic between service network,our partner network
#    and the syslog server on the internal network
iptables -A FORWARD -i $SERVICE_IF -p udp -s $SERVICE_NET -d $SYSLOG \
      --dport 514 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $PARTNER_IF -p udp -s $SERVICE_NET -d $SYSLOG \
      --dport 514 -m state --state NEW -j ACCEPT


# Accept syslog messages from firewall to syslog server
iptables -A OUTPUT -o $INTERNAL_IF -p udp -s $INTERNAL_IP -d $SYSLOG \
      --dport 514 -m state --state NEW -j ACCEPT


# Accept ssh traffic from any internal workstation to any of our networks
iptables -A FORWARD -i $INTERNAL_IF -p tcp -s $INTERNAL_NET -d $SERVICE_NET \
      --dport 22 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $INTERNAL_IF -p tcp -s $INTERNAL_NET -d $PARTNER_NET \
      --dport 22 -m state --state NEW -j ACCEPT

# Accept ssh to firewall from internal network for administration
iptables -A INPUT -i $INTERNAL_IF -p tcp -s $INTERNAL_NET -d $INTERNAL_IP \
      --dport 22 -m state --state NEW -j ACCEPT


# Accept smtp between Internet and mail relay, and between mail relay and mail server
iptables -A FORWARD -i $UPLINK_IF -p tcp -d $MREL_PRIV --dport 25 \
      -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $SERVICE_IF -o $UPLINK_IF -p tcp -s $MREL_PRIV \
      --dport 25 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $SERVICE_IF -p tcp -s $MREL_PRIV -d $MSERV --dport 25 \
      -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $INTERNAL_IF -p tcp -s $MSERV -d $MREL_PRIV \
      --dport 25 -m state --state NEW -j ACCEPT

# Reject any other traffic initiated from our networks to not wait for time-out
iptables -A FORWARD -i $SERVICE_IF -j REJECT
iptables -A FORWARD -i $PARTNER_IF -j REJECT
iptables -A FORWARD -i $INTERNAL_IF -j REJECT
```

(Much help building firewall rules from [11])

As we mentioned, our firewall rules start off with setting our variables, such as the interfaces and IP addresses of the firewall, and the IP addresses of the machines that will send traffic through the firewall. We also have the IP addresses of the stratum-2 ntp servers we are going to use and the IP addresses of the networks of our partners.

Next we set several kernel options on our system. As mentioned in the comments, we drop ICMP broadcasts, source-routed packets, ICMP redirect packets, and spoofed packets that came in from the wrong interface and would not be routed out that interface based on our routing tables. We also set our firewall to not create ICMP redirect packets. As all traffic between our networks goes through this firewall, we shouldn't need to redirect traffic to or from any other routers.

Next we initialize our firewall rules by deleting any previous rules. We'll start with a clean slate, and then build our rule set using this script. We clear the rules for all of the chains in all three tables. We then specifically accept any traffic going through the loopback interface of this firewall. This way, if any network application needs to talk to itself, it will be able to use the loopback interface and will not be blocked. As we mentioned before, this traffic should never go to layer 1, the Ethernet adapter or wire. Finally we set the default policy of all of the chains of all three tables. We set the filter chains to have a default drop policy. This way, any traffic that is not explicitly allowed by a filter will be dropped. We set the default policies of the nat chains and mangle chains to be accept. This way, if a machine does not have a specific nat rule or mangle rule, such as the machines in our internal network have no nat rules, they will still be passed through.

Next, we drop any packets that have illegal TCP flag settings. This will help protect our networks from different attacks, such as scans or floods. We are logging all of this traffic since it is suspicious. We will be able to tell who is sending this improper traffic and decide to block their IP address completely at the router.

Next, we allow any connections that have existing state in the state table. We do this with the "-m state" option (match on state), and set it to match on traffic that already has established or related state. Related is for related traffic, such as ICMP control messages spawned from an established connection. Since iptables compares rules in order, we put this near the top to avoid latency of traffic for established connections.

Next, we configure our NAT rules. We configure the destination NAT to translate our public IP addresses to the appropriate private IP addresses, for our service network servers. This allows connections from the Internet into our servers. This is done in the prerouting chain, so this translation is done as soon as the packets are received by the firewall, before it compares the traffic to its routing table. We then configure our source NAT to translate the reverse: our private IP addresses to the appropriate public IP addresses. This allows outgoing traffic from our servers. This is done in the postrouting chain, so this translation is done very last, after it has made its routing decisions but just before sending the traffic on the wire. We configure the DNAT and SNAT for all of our servers on the service network and partner network that will be accessed by users on the Internet.

Next, we copy all of our rules from the router. The router should stop this traffic, but we are adding another layer of defense just in case any packets manage to sneak through our router. Again, we are logging all of this traffic before we drop it so we can monitor any suspicious activity. Also, these rules are blocking this traffic on all of the firewall's interfaces, not just the uplink to the Internet. This will help protect us so that our own hosts cannot send spoofed traffic through the firewall. This raises a complication, however. Our router is blocking all traffic from any 192.168 network. However, our networks are using IP addresses in that range. If our firewall blocked all traffic from this range on all interfaces, none of our hosts would be able to talk to our other networks. Since we've decided to use this range for our private networks, we will need to change this rule on our firewall. Instead, we have changed the rule to block all traffic from 192.168 that comes from the uplink interface. This would include traffic from our router. We are then dropping any traffic on any interface that is not in the 192.168.0-3.* range. All traffic from any of our hosts should be in this range, so we should be able to drop any traffic that is not in this range. At the end of this list, just as we did in the router, we will have a list of all IP addresses that have previously attacked our network.

Next we will specifically drop some specific traffic that we do not want into our network. Some of this traffic may have been dropped by the default drop rule, but we are putting these in to specifically log the traffic and to make sure that it is explicitly dropped. We are dropping any traffic that has an IP address of all 1's or all 0's (source broadcast or destination broadcast). We are also dropping any broadcast or network address sent to our private networks. Again we are logging this traffic so we can review it.

Finally we are ready to add our rules for traffic we want to allow. We have just finished writing all of the rules that will drop traffic that we are sure we don't want into our networks. This included traffic from invalid IP addresses, traffic with invalid TCP flags set, and broadcast or network address traffic. We will order our accept rules in order of how latency will affect the traffic. Since iptables compares each rule in order, and it will take more time to match against a rule near the end of the list, we want to move real-time traffic higher in priority so it matches sooner. Also, since we are using stateful filtering, and have a rule to match our state table near the top of the rules list, we only need to prioritize initial connections and new traffic. In other words, web traffic, which is real time and the user expects a response as soon as they click their mouse,

will be given higher priority than mail traffic, which is acceptable to be delayed for a few seconds. Finally, we will be concerned about customer traffic before suppliers and partners, and then our own employees will get last priority. After these rules are used, we will be able to monitor each rules match rate, and possibly reorder the rules for better performance, while still following these guidelines.

First we will allow web traffic to our customer web server and supplier web server, and finally our partner web server. We allow http and https, port 80 and 443 respectively, on the customer and supplier web server, but only https on the partner web server. We also have a rule for each of our partners' networks to be able to connect to the partner web server. Then we will allow the database request traffic between the web servers and the database server. Next we will allow DNS request into or out of our service network. Since we are configuring our DNS server to keep all requests less than 512 bytes, we can limit the communication to only use the udp protocol. This will protect us from outsiders trying to do a zone transfer on our DNS server. We can limit both of our DNS servers to only allow udp connections. However, since we do not know how other DNS administrators may have configured their servers, we need to allow both tcp and udp protocol traffic for outgoing DNS requests. Next we will allow web traffic out from our designated web client machine, ntp2, both http and https.

Next we allow ssh traffic from our partners' networks to our partner ssh server. Again, there is a rule for each of our partners' networks. Next we will allow traffic from our ntp servers to the stratum-2 ntp servers. There is a rule for each of the four connections. Next we allow connections from our partner network and internal network to our two ntp servers. There are two connections for each network, one for each server. Next we allow ntp traffic to be initiated from the firewall to the two ntp servers; again one rule for each ntp server. Next we allow syslog traffic from our service network and partner network to the syslog server on the internal network. Next we allow syslog traffic to come from the firewall to the syslog server.

Next we allow ssh traffic to come from our internal network and go to the service network, the partner network, and the firewall itself. Finally, we allow mail traffic to come from the Internet to our mail relay server, to start from the mail relay server to the Internet, to start from the mail relay server to the mail server on the internal network, and to start from the mail server to the mail relay. We have put these rules last since they are the least time sensitive, and will not cause problems if it is delayed by a few seconds.

Finally, we set three rules to reject any other traffic that is initiated from our three private networks. Instead of dropping traffic silently, this will return a packet with the ACK RST flags set. This will tell the client host that this particular connection is not allowed immediately, rather than waiting for a TCP timeout. This will also take care of any latency caused by AUTH/Identd service which tries to authenticate users for connections. Some services, such as smtp, try to use this to authenticate the connection. We won't be using this, and rather than having users wait for several minutes for a timeout to forward mail, the connection will get its reset packet and will immediately send the mail. It does this because the client assumes it's talking to the server, not the firewall, and the server tells the client that it's not running that service. However, we are configuring the firewall to emulate this behavior for our internal networks. We are not doing this for our uplink interface because we do not want to give out any connection information to Internet users. Instead we will let the traffic silently drop with the default drop policy.

When considering how to test the firewall, we want to test a baseline and then test the specific rules. For instance, to test dropping rules, we should test traffic without any firewall drop rules, and then test with the drop rules. We should then check to make sure that the traffic was blocked, and can check that it was dropped by the rule we were expecting. For allow rules, we would do the opposite. We would block traffic, and then add the allow rule to make sure the traffic passes. We should then make sure the expected rule was used to allow the traffic to pass.

For instance, here are three rules taken from our firewall rules above:

```
iptables -t nat -A PREROUTING -i $UPLINK_IF -d $CWEB_PUB -j DNAT \
      --to-destination $CWEB_PRIV
iptables -A FORWARD -i $INTERNAL_IF -p tcp -s $INTERNAL_NET -d $SERVICE_NET \
      --dport 22 -m state --state NEW -j ACCEPT
iptables -A FORWARD -i $UPLINK_IF -s 192.168.0.0/16 -j DROP
```

The first rule is one of our NAT rules. We want to make sure that our NAT rules are working, or else the other rules may not work correctly, since they depend on the prerouting NAT rules to work successfully first. We could test this by running tcpdump on both sides of the firewall. We could then initiate traffic from the outside, noting that the destination has the public address. We can then watch the traffic on the other side of the firewall. We should see the exact same traffic, except the destination should now be the private address. This will show us that our NAT rule is working correctly.

The second rule is an accept rule. As we said, we should first remove this rule from the firewall and make sure ssh traffic is being blocked by the firewall. This would be our baseline. If it is succeeding without this rule, then our rules are not working as expected, and we should figure out which rule is letting it pass. We can use tcpdump on both sides of the firewall to make sure that no traffic patterns from these ssh connections is getting through the firewall. After we're sure the firewall is blocking the traffic, we can add the rule, in its appropriate place. Iptables automatically shows you a hit count of each rule with the "-L <chain> -v" option. We can use this to make sure the traffic is matching on the expected rule. We can then run tcpdump on both sides of the firewall, and try connecting with ssh. We should be able to see the traffic on both sides of the firewall, properly NATed if appropriate (which is not in this case). We can then check the match count of this rule in iptables to make sure that we've matched by the same number of connections we've made.

The third rule is a drop rule. This will be similar to testing the accept rule, but slightly different. Again we want a baseline, so we will turn off all rules of the firewall, or at least apply a default accept rule. We want to send traffic to the uplink interface of the firewall, with a source address in the 192.168.*.* range. We want to put tcpdump on both sides of the firewall, and make sure that the traffic is passing through the firewall. Then we want to apply our rules and test again. This time, we should not see the traffic on the inside of the firewall, again testing with tcpdump. We can again check the match count of this specific rule, and make sure it has incremented by the number of connection attempts we have made.

SSH (VPN solution)

As previously explained, we are using OpenSSH as our solution for a connection between the partners and GIAC. We will configure this machine on an isolated network, and we have protected this network with the firewall rules stated above. Basically, we are blocking all traffic to this network that does not come from a partner's network or our internal network. Also, we're only allow ssh traffic to the ssh server and secure channel web traffic to the partner web server. We're also strictly limiting the allowed connectivity from machines in this network to machines outside of this network.

The ssh server is called sshd. We will configure sshd on this server to better protect the machine itself as well. As an extra layer of protection, we will configure sshd to use tcpwrappers and configure tcpwrappers with similar rules as our firewall. Tcpwrappers is a program that allows you to manage access rules to different network services on a given machine. A given service needs to be compiled to use tcpwrappers, so we will want to make sure that our version of sshd is compiled to use tcpwrappers. We will then configure tcpwrappers to only allow ssh connections from the partners' networks and from our internal network.

This is the configuration file for sshd. On our installation, the file is configured to be located at /etc/ssh/sshd_config.

```
# $OpenBSD: sshd_config,v 1.42 2001/09/20 20:57:51 mouring Exp $ # This sshd was compiled with PATH=/usr/bin:/bin:/usr/sbin:/sbin
# This is the sshd server system-wide configuration file. See sshd(8) # for more information.

Port 22
Protocol 2
#ListenAddress 0.0.0.0
#ListenAddress ::

# HostKey for protocol version 1
```

```
# HostKey /etc/ssh/ssh_host_key
# HostKeys for protocol version 2
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key

# Lifetime and size of ephemeral version 1 server key
KeyRegenerationInterval 3600
ServerKeyBits 768

# Logging
SyslogFacility AUTH
LogLevel INFO
#obsoletes QuietMode and FascistLogging

# Authentication:

LoginGraceTime 600
PermitRootLogin no
StrictModes yes

RSAAuthentication yes
PubkeyAuthentication yes
#AuthorizedKeysFile %h/.ssh/authorized_keys

# rhosts authentication should not be used
RhostsAuthentication no
# Don't read the user's ~/.rhosts and ~/.shosts files
IgnoreRhosts yes
# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
RhostsRSAAuthentication no
# similar for protocol version 2
HostbasedAuthentication no
# Uncomment if you don't trust ~/.ssh/known_hosts for RhostsRSAAuthentication
#IgnoreUserKnownHosts yes

# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication yes
PermitEmptyPasswords no

# Uncomment to disable s/key passwords
ChallengeResponseAuthentication no

# Uncomment to enable PAM keyboard-interactive authentication
# Warning: enabling this may bypass the setting of 'PasswordAuthentication'
#PAMAuthenticationViaKbdInt yes

# To change Kerberos options
#KerberosAuthentication no
#KerberosOrLocalPasswd yes
#AFSTokenPassing no
#KerberosTicketCleanup no

# Kerberos TGT Passing does only work with the AFS kaserver
#KerberosTgtPassing yes

X11Forwarding no
X11DisplayOffset 10
PrintMotd yes
#PrintLastLog no
KeepAlive yes
#UseLogin no

#MaxStartups 10:30:60
#Banner /etc/issue.net
#ReverseMappingCheck yes

AllowTcpForwarding no

Subsystem sftp /usr/lib/ssh/sftp-server
```

Basically, we are mostly using the default configuration. However, we have editted the following parameters:

```
Protocol 2

# HostKey /etc/ssh/ssh_host_key
```

These two options are used for supporting different ssh versions. The first sets which versions of the ssh protocol we accept. The default is to accept both versions 2 and 1. We have limited so that only version 2 can be used. Version 1 is considered to be less secure, and many vulnerabilities have been found with older releases of version 1. Plus it uses weaker encryption methods (DES) than ssh version 2. We are going to avoid it altogether. The second line tells ssh where the ssh version 1 key is located. Since we are not allowing ssh version 1, we have decided to comment this line out.

```
PermitRootLogin no
```

This option tells ssh if it should allow direct logins to the root account. The default is to accept these logins. We have decided to not allow direct logins as root. This way, administrators will have to log in with their own account, and then switch to the root user after logging in. This gives a level of accountability, as you can tell who logged into the machine and then switched to the root user.

```
AllowTcpForwarding no
```

This option disables TCP forwarding of ports. Since users have shell access on the machine, they would be able to forward other network services through ssh, since it is allowed through the firewall. To discourage this, we are not allowing TCP port forwarding on this machine.

This is the tcpwrappers hosts.allow and hosts.deny files (located in /etc/). Tcpwrappers first checks for a match in hosts.allow. If it exists, it allows a connection. If not, it then looks at the hosts.deny. If there is a match, the server does not allow a connection. If it does not match, tcpwrappers allows the connection, as it has a default allow

policy. We will override this by telling the hosts.deny file to deny any connections. This way, if it does not match in the hosts.allow file, the connection will be dropped.

hosts.allow:

```
# See tcpd(8) and hosts_access(5) for a description.

#ALL EXCEPT in.fingerd, in.identd : ALL : spawn (safe_finger -l @%h 2>&1| \
# /bin/mail -s "%d-%h %u" root) &

sshd : 192.168.3.,1.130.0.,1.140.1.,1.150.2.
```

As we mentioned, the hosts.allow file is allowing connections to the sshd server that come from our internal network, and our 3 partners' assumed public networks (as assumed in our firewall rules). Note the IP addresses are comma delimited, and only have 3 octets and end in a ".". This is the wildcard syntax for tcpwrappers, and it means that all machines in the given network are acceptable. For instance, 192.168.3. means all hosts in the 192.168.3.0/24 network.

hosts.deny:

```
# See tcpd(8) and hosts_access(5) for a description.

sshd : ALL : spawn (echo `date` \
     ': Access granted to %c[%a] for %s[%A].' >> /var/log/tcpwrappers)
```

As we stated above, we are using a default deny rule in the hosts.deny. If any traffic is not explicitly matched in the hosts.allow file, it will be dropped. We are also logging any traffic that matches this deny rule so we can monitor any possible attempts at unauthorized access. The various "expansions" (variables) that are used are: %c = client information, %a = client address, %s = server information (sshd), and %A = server address.

Finally, as we stated, we will give the partners access to a partner web server which will be located on the Partner Network. Again, the firewall will block access all access to this machine (and network) unless it comes from our internal network or from one of our partners' networks. Also, this web server will only accept secure channel web traffic through the standard https protocol and port 443.

Intrusion Detection System

We will place intrusion detection systems on our service and partner networks. We will be using Snort, which compares network traffic to known signatures, and logs and alerts whenever there is a match on a given signature. At first we will be using the default rules that come with snort. However, as we monitor traffic and learn new patters, we will be able to write our own rules to alert us when additional patterns match. At first we will be using snort with syslog. This way all of the logs will be pushed to a different server immediately. However, if it seems that snort or syslog is not fast enough to keep up with our network traffic, we can switch to the "high-performance configuration" which stores the snort data in tcpdump binary format to be filtered later.

The particular command that we will use with snort will be:

/usr/local/bin/snort -c /etc/snort/snort.conf -s -D

This executes snort, using the appropriate config file with the -c option, sending it to syslog with the -s option, and running it as a daemon process in the background with the -D option.[12]

Assessment 3

Security Audit

Audit Plan

After implementing the security policies, the next step is to properly audit the policies to make sure they are appropriate, correct, and working. Put simply, we want to make sure the traffic we want to allow can get through, and the traffic we want to stop cannot. Specifically, we will worry most about the primary firewall. However, we should also be concerned about the router, about the servers and hosts, services running on those hosts, and even file/user permissions. Also, we shouldn't forget physical security. We are building the firewall to have strict rules about traffic coming from the Internet, and we are being less strict about allowing access between our networks, as more access is needed for our standard business operations. However, this means that a physical security breach can give someone more access to our networks than they should have. Whether it's someone logging onto one of our hosts, or someone having access to our server room, or someone connecting their laptop to our network, we need to be concerned that an unauthorized user may be able to sneak into our network and avoid some of the protections that we've installed. For this reason, we need to be concerned about our users connecting wireless devices to our network, as this may give an opportunity to secretly connect to our network.

As mentioned, we are going to specifically plan and conduct an audit of our primary firewall. We want to make sure that valid network traffic actually works through the firewall, and that unwanted network traffic is properly blocked. So, there are two general types of traffic that we need to worry about: the traffic that we can expect (be it valid or suspicious) and the traffic that we cannot expect. We can test the former by testing every rule in our firewall, as we are obviously expecting these types of traffic. We can test the latter, to an extent, by using programs that will test for vulnerabilities that we didn't think of. For instance, a good tool to use is nmap. Nmap maps your networks to find hosts, and then it port scans any hosts that it finds. We will use this tool to test unexpected network traffic, which should be dropped by the firewall's default deny policies.

First, we should plan our firewall rules test. For this test to work completely, the firewall should be disconnected from the Internet. We will be allowing more traffic through the firewall than we want Internet users to have access to. So, this test must either be done before installation or while our networks are offline. It will probably take a couple of days to perform these tests, so we'll probably want to do this before installation. We will test each of our rules, one at a time, by adding a new rule to the firewall and testing it. To be sure our test is accurate, we will make sure all the rules that precede the rule in question are actually loaded during the test. We will load all rules before the rule in question, and then send appropriate traffic at the firewall. The behavior should be the opposite of what we want, since we haven't loaded the rule in question yet. After making sure we're seeing the opposite behavior, we should load the rule in question, and run the test again. This time we should see the correct behavior. To test the traffic on both sides of the firewall, we will run tcpdump on both sides. This will show us if traffic is showing up on both sides of the firewall and is being routed, or if it is being dropped and does not show up on one side. We will also check the match count of the rules in the firewall to make sure the appropriate rule was matched. For deny rules, we will set the default policy to be accept and run the test. Then we will add the rule, and make sure the traffic is blocked, though the default policy is accept. For accept rules, we will do the opposite. We will set the default policy to deny, and make sure the appropriate traffic is blocked. Then we will add our rule and test again. This time the traffic should pass through, even with the default deny policy. We can then check the match count of the rules to make sure we're matching on the appropriate rule. Obviously we will not be able to test all possible patterns that would match a rule, for instance, all hosts in a network listed with wildcards. We will choose a suitable test that is in the appropriate range, and test it against the default policy when the rule is missing, and then again with the rule added. If the traffic is passed and blocked appropriately,

and we matched on the appropriate rule, we will assume the other hosts in the range would follow the same pattern. Because rules are compared in order, as long as we test each rule individually, with all prior rules already loaded and tested, we should be able to prove that the complete rule set is working appropriately and allowing appropriate traffic through.

The other part we should test is traffic that we do not want to allow and may not expect. We will use nmap to attempt to find information or holes through our firewall. If we want to see what protection the firewall gives us, we could run nmap on our network with only the NAT/routing of the firewall turned on, and accepting all traffic. Then we could run nmap with all of our rules, and compare the results. The first part should be done while not connected to the Internet, since our firewall will not be blocking any malicious traffic. However, the second part can be done while all of the rules are installed and while the firewall is in full service. However, depending on how aggressive our scanning is, we need to be careful we don't throttle our network performance. We should run these tests with the IDS systems installed in their proper places as well. This way, we can test that the IDS systems are functioning properly at the same time. The IDS system should definitely alert on a port scan from nmap, so we should see these alerts if the IDS is working properly. Also, we also want to test the firewall from all of the interfaces, not just the uplink. We will want to test traffic between our networks and traffic from our networks to the Internet. This way we are sure that our firewall is protecting each of our networks separately, in case one of our machines is compromised.

Audit

As mentioned, first we will test our firewall rules. We will disconnect our firewall from the router, and instead connect two hosts on the uplink interface of the firewall. We will run tcpdump on one of the machines and use the other machine to create appropriate traffic to test our rules. We will want a host on each of our internal networks running tcpdump as well, to make sure traffic is not routed to the wrong network. We will want to make sure the system time on each of these machines is synchronized to make it easier to follow the tcpdump output among these tcpdump hosts. We will run the following command on each of the tcpdump hosts.

tcpdump -nn host <tester_host_ip> -w tcpdumptest#.out

This will execute tcpdump, will print IP addresses and port numbers, instead of hostnames and known port service names, and will filter traffic and only store traffic with a source or destination IP address of our testing host on the uplink interface. We will also write the output in tcpdump binary format to the file tcpdumptest#.out, where # is the number of the test we are running. We will increment this number for each test we run. We are using the write binary option for performance of tcpdump; we want the sniffer to drop as few packets as possible. Afterward the test, we will execute the following commands on each of the hosts.

tcpdump -nn host <tester_host_ip> -r tcpdumptest#.out

This will read in our binary file and output readable text.

As we mentioned, we will test each rule, one at a time, first testing the baseline and then adding the rule to our previously tested rules, and then testing our rule. Our first tests are to check for illegal tcp flag settings. For this, we will use nmap to generate appropriate packets. Our first rule is for packets with all tcp flags turned off. We will set our firewall to have a default allow policy on all chains of the filter tables.

```
iptables --policy INPUT ACCEPT
iptables --policy OUTPUT ACCEPT
iptables --policy FORWARD ACCEPT
```

We also make sure all of the rules that come before this rule that we are testing, are actually loaded, in order, in the firewall. However, we do not want the rule we are testing loaded yet. We then make sure tcpdump is running on all of the tcpdump hosts, and use nmap to send traffic with all tcp flags blank.

nmap -sN -P0 -p 80 '1.129.1.2'

With this command, we see traffic on both sides of the firewall, from our testing host, to our customer web server, and the traffic should all have all TCP flags unset. From the tcpdump host on the uplink network, we see

```
15:35:41.170000 192.168.0.3.51783 > 1.129.1.2.http: . 0:0(0) win 1024
15:35:41.170000 1.129.1.2.http > 192.168.0.3.51783: R 0:0(0) ack 0 win 0 (DF)
```

However, From the tcpdump host on the service network we don't see any traffic. This is strange since we have no rules blocking any traffic. We must assume that iptables, the firewall's stack is not passing this traffic because it has illegal flag settings. We are getting a reset back, but it's not coming from the real host, so it must be coming from the firewall directly.

We then add the rule in question to the firewall.

```
iptables -A FORWARD -p tcp --tcp-flags ALL NONE -j LOG \
     --log-prefix "SCAN? BadTCPFlags: NONE"
iptables -A FORWARD -p tcp --tcp-flags ALL NONE -j DROP
```

We add the log rule as well, just so we can test the logging. We then run the exact same nmap test as above. This time we see the exact same behavior as before. This makes sense because it seems that something is replying to this traffic before iptables can nat and route it. That is why we never see the traffic on the other side of the firewall. It turns out, however, that when we add the following two INPUT rules, we see that we are blocking the traffic.

```
iptables -A INPUT -p tcp --tcp-flags ALL NONE -j LOG \
      --log-prefix "SCAN? BadTCPFlags: NONE"
iptables -A INPUT -p tcp --tcp-flags ALL NONE -j DROP
```

After adding these rules, and running the test again, we see the following from the tcpdump host on the uplink interface.

```
15:49:41.650000 192.168.0.3.45796 > 1.129.1.2.http: . 0:0(0) win 3072
15:49:47.670000 192.168.0.3.45797 > 1.129.1.2.http: . 0:0(0) win 3072
```

This time, there was no reset reply, and nmap waited for a time out. After its timeout, nmap assumes the port is open, which is why it is not receiving a reset. It's assuming this because we told it not to ping test the machine first--to assume that the machine was online. To make sure, we then check with the firewall to make sure the traffic matched on the appropriate rule.

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

We look for the appropriate rule, and see that packets have been matched against that rule.

```
pkts    bytes   target  prot    opt     in      out     source          destination
2       80      LOG     tcp     --      *       *       0.0.0.0/0               0.0.0.0/0 tcp flags:0x023F/0x020 LOG flags 0 level 4 prefix `SC
2       80      DROP    tcp     --      *       *       0.0.0.0/0               0.0.0.0/0 tcp flags:0x023F/0x020
```

This shows a little about how we would test INPUT rules, rather than FORWARD rules, which I discuss in more detail below. This also shows an interesting lesson about the

order of the rules. In my original firewall rules, I listed this INPUT rule above the FORWARD rule. However, for explanation sake, I tested the FORWARD rule before the INPUT rule. If I had loaded the INPUT rule first, my baseline test would not have replied a reset as I would have expected. It would be blocked by the firewall, but I wouldn't know why. I would have to look at the rule match count, as above, to notice that my test was actually matching on this INPUT rule though I expect it to pass through the firewall.

We would do similar tests for all of the other tcp-flag rules, adding each one to the growing list and testing it, one at a time.

For IP address filter rules, we will do a similar test. For instance, we will test the filter of traffic from a source of 10.0.0.2, which should be dropped as it is from a private address. To test this, we change the IP address of our testing host to be 10.0.0.3. We also change the IP address of the uplink interface of the firewall and of the tcpdump test machine on this network, so they all have IP addresses in the same network. We use 10.0.0.1 for the firewall interface and 10.0.0.3 for the tcpdump machine. We then run tcpdump on all of our tcpdump test machines and use 10.0.0.2 as the tester_host_ip address. Again, we make sure that the default policy for the filter chains is set to accept, just as above. We then try to connect to our customer web server on port 80 from our test host.

```
telnet 1.129.1.2 80
```

We see traffic from the 10.0.0.2 address on both sides of the firewall. On the uplink network we see

```
17:19:22.880000 10.0.0.2.34865 > 1.129.1.2.http: S 1021566989:1021566989(0) win 5840 <mss 1460,sackOK,timestamp 67653453
0,nop,wscale 0> (DF) [tos 0x10]
17:19:22.890000 1.129.1.2.http > 10.0.0.2.34865: S 1010321853:1010321853(0) ack 1021566990 win 5792 <mss 1460,sackOK,timestamp
59177430 67653453,nop,wscale 0> (DF)
17:19:22.890000 10.0.0.2.34865 > 1.129.1.2.http: . 1:1(0) ack 1 win 5840 <nop,nop,timestamp 67653454 59177430> (DF) [tos 0x10]
17:19:22.890000 10.0.0.2.34865 > 1.129.1.2.http: . 1:1(0) ack 1 win 5840 <nop,nop,timestamp 67653454 59177430,nop,nop, sack 1 {0:1}>
(DF) [tos 0x10]
```

On the service network we see

```
17:19:15.717389 10.0.0.2.34865 > 192.168.1.2.80: S 1021566989:1021566989(0) win 5840 <mss 1460,sackOK,timestamp 67653453
0,nop,wscale 0> (DF) [tos 0x10]
17:19:15.717389 192.168.1.2.80 > 10.0.0.2.34865: S 1010321853:1010321853(0) ack 1021566990 win 5792 <mss 1460,sackOK,timestamp
59177430 67653453,nop,wscale 0> (DF)
17:19:15.727389 10.0.0.2.34865 > 192.168.1.2.80: . 1:1(0) ack 1 win 5840 <nop,nop,timestamp 67653454 59177430> (DF) [tos 0x10]
17:19:15.727389 10.0.0.2.34865 > 192.168.1.2.80: . 1:1(0) ack 1 win 5840 <nop,nop,timestamp 67653454 59177430,nop,nop, sack 1 {0:1}
>(DF) [tos 0x10]
```

We then add our rule to the firewall.

```
iptables -A FORWARD -s 10.0.0.0/8 -j LOG \
      --log-prefix "Spoofed Packet passed router?"
iptables -A FORWARD -s 10.0.0.0/8 -j DROP
```

Again, we add the log rule just to test that the logging is working correctly. We then run telnet from our host again, just as above, and check the tcpdump output of our hosts. This time we see traffic on the uplink network, but do not see traffic on the inside, just as we should. We can tell that the testing server is not getting a reply because of the time of each sent packet--it is waiting for a TCP timeout. The testing host eventually gives a timeout error.

```
17:28:46.280000 10.0.0.2.34866 > 1.129.1.2.http: S 1620279305:1620279305(0) win 5840 <mss 1460,sackOK,timestamp 67709793
0,nop,wscale 0> (DF) [tos 0x10]
17:28:49.280000 10.0.0.2.34866 > 1.129.1.2.http: S 1620279305:1620279305(0) win 5840 <mss 1460,sackOK,timestamp 67710093
0,nop,wscale 0> (DF) [tos 0x10]
17:28:55.280000 10.0.0.2.34866 > 1.129.1.2.http: S 1620279305:1620279305(0) win 5840 <mss 1460,sackOK,timestamp 67710693
0,nop,wscale 0> (DF) [tos 0x10]
17:29:07.280000 10.0.0.2.34866 > 1.129.1.2.http: S 1620279305:1620279305(0) win 5840 <mss 1460,sackOK,timestamp 67711893
0,nop,wscale 0> (DF) [tos 0x10]
17:29:31.280000 10.0.0.2.34866 > 1.129.1.2.http: S 1620279305:1620279305(0) win 5840 <mss 1460,sackOK,timestamp 67714293
0,nop,wscale 0> (DF) [tos 0x10]
17:30:19.280000 10.0.0.2.34866 > 1.129.1.2.http: S 1620279305:1620279305(0) win 5840 <mss 1460,sackOK,timestamp 67719093
0,nop,wscale 0> (DF) [tos 0x10]
```

We finally check the match count on the iptables rules to make sure the appropriate rule matched. We use the exact same list (iptables -L) command as above. We notice that traffic has matched on this rule, so it is working appropriately.

```
pkts    bytes   target  prot    opt     in      out     source          destination
6       360     LOG     all     --      *       *       10.0.0.2        0.0.0.0/0       LOG flags 0 level 4 prefix `Spoofed Packet passed route
6       360     DROP    all     --      *       *       10.0.0.2        0.0.0.0/0
```

Since we've already added all of the previous rules, and this traffic matched on this rule, we can be sure no other rules will handle this type of traffic before this rule. We have to be careful when changing the orders of rules after we've tested the firewall however.

This is how we will test all of our forwarding rules. However, we will have to test our INPUT and OUTPUT rules a little differently. Unfortunately, we cannot test this on both sides of the firewall, because this traffic has a source or destination of the firewall itself. It is not passing this traffic through to another network. So, we will have to test with a single instance of tcpdump running. For instance, take the following rule, which is similar to the one above but is for the INPUT chain instead of the FORWARD chain.

```
iptables -A INPUT -p tcp --tcp-flags ALL NONE -j LOG \
--log-prefix "SCAN? BadTCPFlags: NONE"
iptables -A INPUT -p tcp --tcp-flags ALL NONE -j DROP
```

We setup a similar test, running tcpdump on host on the uplink network and then running the same nmap command as before. However, we need to test this by checking for reply traffic, just as we did above. We notice that without the rule, we get a reset packet in reply from the firewall. When we apply the rule, we should not get a reset packet in reply because the firewall should drop the traffic silently. When we run the test after rule is applied, we see that this is the case. Therefore, we can test the INPUT and OUTPUT rules by checking for traffic or reply traffic on the appropriate network that the traffic should be sent to, or blocked from sending to.

After testing all of the rules of the firewall in this manner, we can feel certain that our rules are working, and allowing and blocking appropriate traffic. Next we can try a brute force attack against our network to see if any additional information is accessible. This would include open ports, or any other traffic that should not be allowed. For this test, we will want to make sure that our IDS systems are configured and running. While doing a brute force test, we will be able to test that the IDS system is working as well.

Again, we will want a baseline. So, time permitting, we want to run a brute force scan without firewall rules and then with firewall rules, and compare the two. Nmap has many different options to test many types of vulnerabilities. We will want to use most of these options to run different tests. However, this is likely to take a very long time

to do a full scan, possibly up to a week. So, this probably is not viable. Instead, we will use the timing option set to "aggressive" which puts maximum timeout times per port and total per host. We also setup the tcpdump hosts on all of the networks to monitor any traffic that does get through the firewall. These are some of the nmap commands we will use to test the network.

```
/usr/bin/nmap -sT -v -T Aggressive -oN nmaplog.001 '1.129.1.0/24' >> nmapscan.out
/usr/bin/nmap -sS -v -T Aggressive -oN nmaplog.002 '1.129.1.0/24' >> nmapscan.out
/usr/bin/nmap -sF -v -T Aggressive -oN nmaplog.003 '1.129.1.0/24' >> nmapscan.out
/usr/bin/nmap -sX -v -T Aggressive -oN nmaplog.004 '1.129.1.0/24' >> nmapscan.out
/usr/bin/nmap -sN -v -T Aggressive -oN nmaplog.005 '1.129.1.0/24' >> nmapscan.out
/usr/bin/nmap -sP -v -T Aggressive -oN nmaplog.006 '1.129.1.0/24' >> nmapscan.out
/usr/bin/nmap -sU -v -T Aggressive -oN nmaplog.007 '1.129.1.0/24' >> nmapscan.out
/usr/bin/nmap -sA -v -T Aggressive -oN nmaplog.009 '1.129.1.0/24' >> nmapscan.out
/usr/bin/nmap -sW -v -T Aggressive -oN nmaplog.010 '1.129.1.0/24' >> nmapscan.out
/usr/bin/nmap -sR -v -T Aggressive -oN nmaplog.011 '1.129.1.0/24' >> nmapscan.out
```

These scans test for TCP connections, SYN scans, FIN scans, Xmas tree scans, Null scans, Ping scans, UDP scans, ACK scans, "Window" scans and "RPC" scans. We have the scans set to verbose mode, set the timing parameters to Aggressive mode, and creates an output log in a readable format. It runs this scan against our entire public IP network and appends any standard output to another log file. We're also interested in using the -PT, -P0, -PS, -PI, and -PB options for all of these scans. These options are used to tell nmap how to identify online hosts before port scanning. Each option uses "tcp ping" on port 80, no ping, SYN connection requests, ICMP echo request, and finally "tcp ping" and ICMP ping type, respectively. Actually using -PB should make -PT and -PI unnecessary, since it uses both of these mechanisms. The interesting one is -P0, as this doesn't check for online hosts. This option does an entire port scan of every possible IP address in the network. Since it will be waiting for many TCP timeouts, this option takes a very long time to test with.

Example output of the nmap scans from above without firewall rules follow. We only see two hosts here, 192.168.1.2 and 192.168.2.8, because we had a limited number of machines actually online during this scan.

nmaplog.001:

```
Interesting ports on (1.129.1.2):
(The 1521 ports scanned but not shown below are in state: closed)
Port State Service
22/tcp open ssh
80/tcp open http

Interesting ports on (1.129.1.8):
(The 1522 ports scanned but not shown below are in state: closed)
Port State Service
22/tcp open ssh

# Nmap run completed at Fri Mar 29 20:30:25 2002 -- 256 IP addresses (2 hosts up) scanned in 25 seconds
```

Most of the scans have basically the same output. The only one that is very different is nmaplog.010, the Window scan. It was able to tell these hosts were online, but found no available open ports.

From running these scans, we can look at our IDS alerts to see if it found any suspicious activity. Since we were running so many mappings our IDS should plenty of alerts. In fact, it does. We have over 160000 alerts, mainly for FIN scans and XMAS scans. Below is some sample output of the alerts.

```
[**] [111:8:1] spp_stream4: STEALTH ACTIVITY (FIN scan) detection [**]
03/28-22:00:01.427389 192.168.0.3:59068 -> 192.168.2.8:817
TCP TTL:40 TOS:0x0 ID:534 IpLen:20 DgmLen:40
*******F Seq: 0x0 Ack: 0x0 Win: 0x800 TcpLen: 20

[**] [111:10:1] spp_stream4: STEALTH ACTIVITY (nmap XMAS scan) detection [**]
03/28-22:00:08.217389 192.168.0.3:33934 -> 192.168.1.2:1466
TCP TTL:57 TOS:0x0 ID:33659 IpLen:20 DgmLen:40
**U*P**F Seq: 0x0 Ack: 0x0 Win: 0xC00 TcpLen: 20 UrgPtr: 0x0
```

We also see plenty of traffic on our tcpdump outputs of our service tcpdump machine. Here's a sample of the output of the beginning of a scan.

```
00:24:09.097389 192.168.0.3 > 192.168.1.2: icmp: echo request
00:24:09.097389 192.168.1.2 > 192.168.0.3: icmp: echo reply (DF)
00:24:09.097389 192.168.0.3.39501 > 192.168.1.2.80: . 1403519059:1403519059(0) ack 2615968728 win 4096
00:24:09.097389 192.168.1.2.80 > 192.168.0.3.39501: R 2615968728:2615968728(0) win 0 (DF)
00:24:09.097389 192.168.0.3 > 192.168.2.8: icmp: echo request
00:24:09.097389 192.168.2.8 > 192.168.0.3: icmp: echo reply (DF)
00:24:09.097389 192.168.0.3.39501 > 192.168.2.8.80: . 3647996227:3647996227(0) ack 94070546 win 4096
00:24:09.097389 192.168.2.8.80 > 192.168.0.3.39501: R 94070546:94070546(0) win 0 (DF)
00:24:09.747389 192.168.0.3.53364 > 192.168.1.2.306: S 2602871881:2602871881(0) win 5840 <mss 1460,sackOK,timestamp 18364180 0,nop,wscale 0> (DF)
00:24:09.747389 192.168.1.2.306 > 192.168.0.3.53364: R 0:0(0) ack 2602871882 win 0 (DF)
00:24:09.747389 192.168.0.3.53365 > 192.168.1.2.1552: S 2611842738:2611842738(0) win 5840 <mss 1460,sackOK,timestamp 18364180 0,nop,wscale 0> (DF)
00:24:09.747389 192.168.1.2.1552 > 192.168.0.3.53365: R 0:0(0) ack 2611842739 win 0 (DF)
00:24:09.747389 192.168.0.3.53366 > 192.168.1.2.342: S 2597131227:2597131227(0) win 5840 <mss 1460,sackOK,timestamp 18364180 0,nop,wscale 0> (DF)
00:24:09.747389 192.168.1.2.342 > 192.168.0.3.53366: R 0:0(0) ack 2597131228 win 0 (DF)
00:24:09.747389 192.168.0.3.53367 > 192.168.1.2.71: S 2597475493:2597475493(0) win 5840 <mss 1460,sackOK,timestamp 18364180 0,nop,wscale 0> (DF)
00:24:09.747389 192.168.1.2.71 > 192.168.0.3.53367: R 0:0(0) ack 2597475494 win 0 (DF)
```

First nmap mapped the network to find available hosts, and then it started port scanning those hosts.

After installing the firewall rules, we get this output from the same nmap commands as above. We changed the testing machine's IP address to be a non-private IP range. Since we've already assumed that the 1.*.*.* network address space has been assigned, and we are not blocking this range, we assigned this host an IP address of 1.128.0.3, to emulate it having a public IP address.

nmaplog.001:

Skipping host (1.129.1.2) due to host timeout
Nmap run completed at Sun Mar 31 17:54:00 2002 -- 256 IP addresses (1 host up) scanned in 326 seconds

We got the following message from standard out, which was appended to the nmapscans.out file:

Host (1.129.1.2) appears to be up ... good.
Initiating TCP connect() scan against (1.129.1.2)
Adding TCP port 80 (state open).
Skipping host (1.129.1.2) due to host timeout

Note two interesting things from this output. One is that it took much longer for the scan to run with the firewall in place. It took about 25 seconds to port scan the network without a firewall. It took more than 326 seconds to port scan with the firewall, and it quit on the host that it found because of our maximum timeout per host setting. This is good because it makes it much harder for attackers to scan our network, since it takes so much more time. The other thing to notice is that this particular scan only found one host, not both. This is because this particular scan first maps our network with ICMP and a TCP connect to port 80. Since we are blocking ICMP traffic at the firewall, and we are blocking port 80 traffic to our SSH server, this scan didn't even know the SSH server existed. Other scans will be able to find hosts that don't allow web traffic, but those scans will be harder to run and will take much more time. In fact, none of our scans found the other server because of our rule limiting traffic towards the SSH server to only be allowed from the partners' networks only. All of the scans either found 0 hosts up, 1 host up, or 256 hosts up. This is very limited information for our attackers.

Our IDS system did not notice anything suspicious on our service network because our firewall was doing such a good job of protecting it. There wasn't enough traffic on the wire to trip the IDS and cause it to send an alert. Basically, in this particular scan, attackers would be able to find out that our customer web server allowed web traffic. Since this is probably not a secret, and our customer web server is completely public, and we probably advertise it, this shouldn't be a problem. We just have to make sure that we've taken appropriate precautions to avoid giving out any other information, hence this scan.

Finally, here is a sample of the tcpdump output taken during these scans.

```
17:52:30.057389 1.128.0.3.42955 > 192.168.1.2.80: . 3313500243:3313500243(0) ack 3775518897 win 1024
17:52:30.057389 192.168.1.2.80 > 1.128.0.3.42955: R 3775518897:3775518897(0) win 0 (DF)
17:53:01.067389 1.128.0.3.60993 > 192.168.1.2.80: S 3295658277:3295658277(0) win 5840 <mss 1460,sackOK,timestamp 33296721
0,nop,wscale 0> (DF)
17:53:01.067389 192.168.1.2.80 > 1.128.0.3.60993: S 3282934340:3282934340(0) ack 3295658278 win 5792 <mss 1460,sackOK,timestamp
24819964 33296721,nop,wscale 0> (DF)
17:53:01.067389 1.128.0.3.60993 > 192.168.1.2.80: . 1:1(0) ack 1 win 5840 <nop,nop,timestamp 33296721 24819964> (DF)
17:53:01.067389 1.128.0.3.60993 > 192.168.1.2.80: . 1:1(0) ack 1 win 5840 <nop,nop,timestamp 33296721 24819964,nop,nop, sack 1 {0:1}
> (DF)
17:53:01.067389 1.128.0.3.60993 > 192.168.1.2.80: R 1:1(0) ack 1 win 5840 <nop,nop,timestamp 33296721 24819964> (DF)
17:53:36.907389 1.128.0.3.34143 > 192.168.1.2.443: S 3353299257:3353299257(0) win 5840 <mss 1460,sackOK,timestamp 33300305
0,nop,wscale 0>(DF)
17:53:36.907389 192.168.1.2.443 > 1.128.0.3.34143: R 0:0(0) ack 3353299258 win 0 (DF)
17:53:45.777389 1.128.0.3.48573 > 192.168.1.2.80: . 723517523:723517523(0) ack 3954475693 win 2048
17:53:45.777389 192.168.1.2.80 > 1.128.0.3.48573: R 3954475693:3954475693(0) win 0 (DF)
17:54:30.827389 1.128.0.3.48553 > 192.168.1.2.80: S 627502500:627502500(0) win 2048
17:54:30.827389 192.168.1.2.80 > 1.128.0.3.48553: S 3395260754:3395260754(0) ack 627502501 win 5840 <mss 1460> (DF)
17:54:30.827389 1.128.0.3.48553 > 192.168.1.2.80: R 627502501:627502501(0) win 0 (DF)
17:55:01.587389 1.128.0.3.61988 > 192.168.1.2.80: . 2314731603:2314731603(0) ack 682492966 win 1024
17:55:01.587389 192.168.1.2.80 > 1.128.0.3.61988: R 682492966:682492966(0) win 0 (DF)
17:56:17.297389 1.128.0.3.36197 > 192.168.1.2.80: . 3339190355:3339190355(0) ack 830298639 win 4096
17:56:17.297389 192.168.1.2.80 > 1.128.0.3.36197: R 830298639:830298639(0) win 0 (DF)
17:57:33.037389 1.128.0.3.59613 > 192.168.1.2.80: . 2663383123:2663383123(0) ack 4019039288 win 4096
17:57:33.037389 192.168.1.2.80 > 1.128.0.3.59613: R 4019039288:4019039288(0) win 0 (DF)
17:58:49.037389 1.128.0.3.47771 > 192.168.1.2.80: . 3680501843:3680501843(0) ack 2222445621 win 1024
17:58:49.037389 192.168.1.2.80 > 1.128.0.3.47771: R 2222445621:2222445621(0) win 0 (DF)
17:58:49.737389 1.128.0.3.41237 > 192.168.1.2.80: . 1728053331:1728053331(0) ack 1336264958 win 2048
17:58:49.737389 192.168.1.2.80 > 1.128.0.3.41237: R 1336264958:1336264958(0) win 0 (DF)
18:00:05.447389 1.128.0.3.52674 > 192.168.1.2.80: . 3975151699:3975151699(0) ack 1255812981 win 2048
18:00:05.447389 192.168.1.2.80 > 1.128.0.3.52674: R 1255812981:1255812981(0) win 0 (DF)
```

Note the only traffic that got through the firewall was traffic directed at port 80 and port 443 of our web server. Everything else was blocked.

Next we have to run similar tests between all of our internal networks and then from our internal networks to the Internet. Of course our non-NATed address should not be able to route traffic to the Internet, but we will make sure the same way we've run these tests, with a tcpdump machine on each end of the firewall and the testing machines generating traffic. After testing this, tcpdump shows that traffic directed to the Internet from a non-NATed source gets reset packets in return.

Evaluation and Outcome

As we've seen through our tests, our firewall is allowing proper traffic and is basically defending our network from malicious traffic. There were two things that I noticed during the audit from the Internet side. One is that our NAT configuration is making it very difficult to learn much about any of our hosts. Since we are NATing our public addresses to selected private addresses, and since we're not allowing source-routed traffic, we have very good control on the traffic that comes from the Internet. We are keeping our real network infrastructure hidden, and only showing the Internet a single network. We keep them from even possibly mapping our entire network as all they can map is the virtual public network that we've created. Not only is there a small element of security through obscurity, but we also have firewall rules to back this up and keep traffic from the Internet out of services and hosts it's supposed to go. Plus, since our firewall does not have a public Internet address, we are keeping Internet users from being able to connect to it directly. This makes it very difficult for them to locate the firewall or know that it exists.

The other interesting thing that I noticed is how much more difficult it is to map a network or scan hosts when you have a firewall dropping packets. With the firewall in place, many of our tests either found no hosts connected or all hosts connected, even though they were not all connected. So, someone trying to map our network would likely think that there were either no hosts connected or 254. Then, when the test tries to run a port scan it needs to time out on each port to decide that it doesn't exist, even if the machine is there. If there were no firewall, the machine would immediately send back a reset packet. But, the firewall silently drops the packet, so the scanner is forced to wait for a time out. Plus, if you're blocking ICMP on your firewall, the only way to tell if a host exists is by scanning it, port by port. So an attacker would have to do a full port scan of all 256 machines to tell what services are listening. As an example, I ran a simple test on my network, once without the firewall rules and once with the firewall rules. The test was "nmap -sT -PT". This scan first maps the network for hosts listening on port 80 and then port scans live hosts with TCP connects. Running this with firewall rules in place took about 10 to 15 times longer than without the firewall rules, because it was waiting for timeouts to move onto the next port. Some tests, such as "nmap -P0" which scans all ports of every host, would take hours, if not days, to complete a scan of a class C network.

Although our architecture seems to be very protected, there are a couple of improvements that could be made. For instance, currently we have default accept policies on the NAT and mangle tables. Although the filter tables should properly filter traffic, it would be an improvement to configure the firewall so we can have default deny policies on the NAT and mangle tables. For instance, we could configure all of our hosts for NAT, even the ones that do not connect to the Internet, and simply use the same private address for both sides of the translation (translate an IP address to itself). Since we'd be using private addresses, we could still block traffic from the uplink interface that use these addresses. But this way we could configure the NAT to have a default deny policy, which may protect the box more incase any security vulnerabilities are found in

iptables itself. We could make a similar configuration to protect the mangle table as well.

Secondly, we may consider the benefits of using a full VPN. Not only does a VPN have more benefits, such as automatically encrypting any traffic between two networks, it also may add another layer of security that we do not have in our current setup. Although we have a fairly secure configuration of ssh and the partner network, the host is still directly connected to the Internet. If there were a misconfiguration in our firewall, such as we allowed the wrong partner address in, some Internet users may have direct access to the server. If they were able to use a vulnerability of ssh, they may be able to compromise the system, since they would be talking directly to it. Basically, we're only using the firewall to protect the service. In general, a user needs to properly authenticate to the ssh service, and we are using tcpwrappers to authenticate based on IP address as well. But, as we said, a single misconfiguration and a vulnerability may allow the machine to become compromised. However, if we added a VPN layer, an attacker would have to rely on a misconfiguration of the firewall, possibly two if we route all post-encapsulated VPN traffic through the firewall again, plus get through the VPN authentication (secret password or signature key) plus break through the service. This simply adds another layer that an attacker would have to worry about.

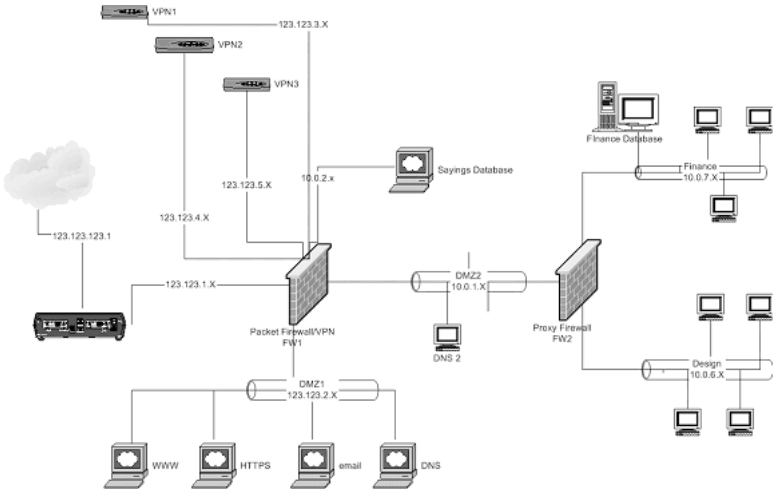
Finally is the possible improvement of the order of the firewall rules and the creation of IDS rules. We will want to watch our traffic patterns and match counts of the firewall to optimize the order of the rules so rules that are more frequently matched appear earlier. Of course we will want to make these changes while checking to make sure we do not break any other expected restriction or conflict with our rules about priority of connections (i.e. customers before employees). We will also want to regularly update any of the IDS rules to have the latest and greatest signatures. We should also consider writing some of our own rules to match any customized concerns we have in our environment.

As we mentioned above there are many other security issues to consider as well, besides the firewall. The router, the hosts, and physical security are all important considerations as well. Plus putting in redundant services or load-balancing services will help us prepare for bursts of data, whether natural, or attempted denial of service attacks. Preparing and planning policies, and implementing them, is the best way of having some confidence in the security of a network architecture.

Assessment 4

Design Under Fire

To help us understand how an attacker may try to attack our own network, we will look at a different network architecture that was designed for a similar environment as our own. For this example, we will look at Susan Caskey's network architecture (http://www.giac.org/practical/Susan_Caskey_GCFW.zip).



Firewall

One thing to be concerned about are attacks on our firewall directly. Since the perimeter firewall is often the main perimeter security device, most, if not all, traffic must go through it. So, if the firewall is attacked directly, it could do much damage to our network infrastructure. For instance, if a denial of service attack were done on it, it would cripple our network connection to the Internet, affecting all customers and employees alike. This could have an irrecoverable cost to our organization. If the firewall were compromised, it's possible, if not probable, that all of our networks have been compromised, since many organizations depend on the firewall to protect their internal hosts.

It seems that there have not been many large security vulnerabilities released recently for the major firewall vendors. Most vulnerabilities depend on some sort of access that an attacker needs to previously have. Others depend on some additional option package to be installed on the firewall. It seems that the firewall vendors are making good firewall products, but their option packages often open vulnerabilities in these firewall products.

Of course, recently, there was a very large security alert about how SNMP vulnerabilities could affect most networking equipment.[14] Many firewalls, routers, and servers are configured for SNMP for monitoring and management. Susan does not mention if she's using SNMP in her network or not. Since she seems to mention all of the services she is planning to use, we can probably guess that she will not use SNMP. However, as an attacker, I would try to see if I could affect her network using SNMP attacks.

Another vulnerability in the Checkpoint FW-1 firewall is a denial of service.[15] This vulnerability is affected by sending a stream of large IP fragments at the firewall. The firewall will use all of it's processing to reassemble the fragments, and since there is a stream of large fragments, it will deny service to other traffic that is trying to pass through the firewall. The CERT document claims that a large stream can affect the vulnerability, and in another place that a stream of large fragments. These are obviously different types of traffic. If it is simply a large stream that will affect this, then using "nmap -f" should help bring down a Checkpoint firewall. This breaks up the scan packets into fragments, to avoid security devices. Performing a fast scan on this server may make it fail. If they mean "large fragments", we could send large packets data, but they will probably be fragmented down to 1500 bytes, which is the standard MTU for Ethernet.

Finally, I notice that the uplink interface of the firewall is on a public routable network. This means that we can send traffic directly to the firewall. The firewall may be configured to block it, but we may be able to perform a flood attack directly at the firewall. If successful, this denial of service would affect traffic performance to all of the hosts behind the firewall. This would obviously be bad for customers and employees, and would definitely cost the organization money. It would probably be more secure to configure the firewall to have private interfaces so Internet traffic cannot be directed at the firewall itself.

Denial of Service

Another thing to be concerned about are denial of service attacks. Unfortunately, they have become very common because they are very effective, and difficult to avoid. For instance, we can always attack a site's web server. Since a site's web server has a public address to get to it, as it's offering a public service, we can find that ip address and

attack the web server directly. In fact, we even know of a particular port that is listening, the http port, port 80. So, we can perform a TCP SYN flood attack on GIAC's web server.[13] We will assume that we already have about 50 compromised cable modem/DSL systems that we took control of before. Since these are cable modem/DSL, we know there is going to be a limit on the amount of throughput we can get out of each machine, and it's probably less than GIAC's bandwidth can handle. However, we do have 50 machines, and we can play some tricks.

Luckily, many cable modem/DSL users leave there machines on all the time. And many of them are at work during the day, so we can use these machines then to attack the web server, and the users will not be suspicious. In fact, since we have control of their machine, we may be able to tell when users are logged in. If any of these machines are Linux machines, or some other version of unix, the job has gotten much easier. We can simply install and use nmap on any of these machines, and let it create the packets for us. In fact, we can use the -D option to specify decoy IP addresses to make it more difficult to tell where the attacks are really coming from. We should add several decoy IP addresses, and we should include some reserved IP addresses. Since GIAC isn't blocking reserved addresses (2.*.*.* for instance) we can use some of these addresses. Better yet, we can also try to find some IP addresses that we know are behind a firewall and use these addresses as decoys. When the GIAC server replies with an ACK packet, the spoofed address' firewall will silently drop the packet, and GIAC's server will wait for a reply for a full TCP timeout. During this time, the connection queue is filling up with other similar requests until this queue is filled. This will cause a denial of service, as the web server will no longer be able to accept requests. Of course, since most cable modem/DSL users are using Windows machines, which is arguably easier to break into than Linux, it won't be quite so easy because nmap isn't ported to Windows, yet. So, we will have to find another packet generator, like http://www.provanet.com or the several located at http://www.tlsecurity.net/windows/Assassement/PacketForging/ . Of course, we could always create our own Win32 application or java application, but then we would have to code, and in the case of java, hope they have java installed. Plus, it's always much easier to just download someone else's program for attacking people.

Since we don't know what type of web server GIAC is using, or the specs of the hardware, we don't know for sure if it would be able to handle a SYN flood from 50 hosts. However, if it has no protection at all, it will probably at least start to slow down, if not become unusable to valid users.

Basically, there are a couple of things that the administrators at GIAC could do to prevent this from happening. One thing that would help may be to block all IANA reserved addresses. However, this won't help for spoofed traffic from networks that are protected by firewalls. For this, we could install some quality of service limits on the router or firewall. Many firewalls have quality of service settings, including the PIX. These settings would set a maximum for the time to wait for an ACK packet to protect against SYN floods. It may also protect for a maximum number of connections in a certain amount of time. For instance, it seems that the real GIAC web site (http://www.giac.org) only allows two connections at a time per IP address, and blocks any more concurrent connections. Of course, this could be used as a denial of service as well, as this will affect people that go through proxies. Perhaps that's why the GIAC web site warns about AOL users having problems with their web site. Yet another cost/benefit compromise, as we have shown many examples of which in this exercise.

Bibliography

1. GIAC GCFW Practical v1.6a
2. SANS Firewalls; "2.4 VPNs and Remote Access", v2.1; Chris Brenton, et al.; pp. 59, 165-73
3. Linux Firewalls, Second Edition; Robert L. Ziegler; pp. 150-2
4. http://www.rsasecurity.com/rsalabs/faq/3-6-6.html
5. http://www.eecis.udel.edu/~mills/ntp/clock2.htm
6. SANS Firewalls; "2.1 TCP/IP for Firewalls", v1.4; J. Novak; pp.'7-19'
7. http://www.orafaq.com/faqnet.htm#VERSIONS
8. SANS Firewalls; "2.3 Firewalls 102: Perimeter Protection and Defense In-Depth", v2.1; CB et al.; p 60
9. RFC:1918, Rekhter, et al.
10. http://www.iana.org/assignments/ipv4-address-space, 2001-12-01
11. Linux Firewalls, Second Edition; Robert L. Ziegler; pp. 111-79
12. http://www.snort.org/docs/writing_rules/
13. http://www.cert.org/advisories/CA-1996-21.html
14. http://www.cert.org/advisories/CA-2002-03.html
15. http://www.kb.cert.org/vuls/id/35958