Global Information Assurance Certification Paper

Benjamin Eason
GCUX 2.0 Option 1
Submitted: May 2, 2004

OpenBSD “Escorting Firewall” Step by Step Guide

Abstract

This step by step guide will help you to both design and deploy your own escorting firewall. An escorting firewall escorts remote access users on trusted networks and revokes their authorization if they violate network security policies. Many organizations allow remote access even without an effective means to secure the network from remote access users. Combining a bridging firewall, signature-based network attack detection, proactive policy compliance auditing, vulnerability scanning, and dynamic response, it is this risk that the “escorting firewall” attempts to mitigate.

Introduction

Written security policies at most organizations deem the act of connecting foreign systems to the network as insecure. However, in the cases of user dial-in and VPN remote access, those same policies permit such insecure actions. Connecting foreign systems to networks for the purpose of remote user access is a risk that managers commonly accept; in many cases it is the only way to collaborate with traveling sales persons and field engineers. Combining a bridging firewall, signature-based network attack detection, proactive policy compliance auditing, vulnerability scanning, and dynamic response, it is this risk that the “escorting firewall” attempts to mitigate.

This step-by-step guide will help you to both design and deploy your own escorting firewall. The difficulty in implementing the system as documented here depends on the complexity of your security policies, network design, and scripting skills. On most small networks, successful implementation of this system will be easily attainable with very Spartan hardware and with limited technical skills. If your remote access server is actually a Cisco PIX 500 series firewall terminating IPSec tunnels, then you absolutely must read this guide.

What is an escorting firewall? The escorting firewall server proactively and reactively enforces two different aspects of network policy: system configuration compliance and acceptable network usage (these two network policies will be repeatedly referenced throughout this paper). Imagine a bouncer at a nightclub that lets people in if they are dressed properly, but kicks them out if they misbehave. Similarly, this special purpose firewall dynamically blocks policy violators from accessing the network. Why not call it a bouncer firewall? The name escorting firewall sounds friendlier; you may call it whatever you like.

As network perimeters become increasingly porous and threats from Internet worms and hackers increase, security administrators are forced to look for creative ways to compensate for weaker perimeter security. Many

organizations have begun hardening internal workstations, deploying host based firewalls, and implementing aggressive patch management systems. These are necessary first steps to avoiding the beetle complex – hard on the outside, soft and gooey on the inside. However, one area that has remained thorny for security strategists is remote user access.

The business benefits of remote access for employees, executives, and contractors are often very high. Administering these remote users and their systems has however always been expensive. Often the connecting systems are not owned by the company and may require configurations that aren't normally supported by IT staff. Additionally, their hours of operation and physical locations may further increase support costs. Not surprisingly, even as the cost benefit analysis of attempting to support remote users is negative, the risk benefit analysis of allowing this type of access is still positive and the business must therefore choose to accept the risk.

For most organizations escorting firewalls should offer a positive cost benefit method for reducing the risks associated with allowing remote user access. Progressive and well-organized security departments might even be able to offer a choice to remote access users; connect from a machine the business has full administrative control over or be escorted while on the network.

Specification

The requirement is to create a mechanism that enforces the local network configuration compliance and acceptable network usage policies for remote access users. This specification defines a specific technical approach to meeting this requirement.

The "escorting firewall" server is a mechanism to reduce the risk of allowing remote users, access to trusted networks. This specification is a high-level description of the functioning of the server under different conditions; specifically, it states what the server is allowed and not allowed to do. In addition to the specification, this section of the practical will cover the design of the server. Many assumptions have been made about the operating environment and site policies. A note will be made where important design decisions are introduced and based on assumptions that may not apply to your environment.

Where possible all descriptions will be environment neutral, however, if a` section is difficult to follow, refer to the example network diagram in appendix A as it may reveal an underlying assumption that hasn't been clearly stated. As shown in the diagram, the escorting firewall bridges the link between the remote access server and rest of the trusted network. If complete transparency is desired, the Nessus daemon can run on a separate system and a third network interface may be used to connect the escorting firewall to an administration network. Look at the second network diagram in appendix A for an example. (This creates transparency because the escorting firewall can be configured without an IP address on all but the special network interface that connects it to the Nessus daemon. The Nessus server then performs scans against new users when they connect and the users only see the IP address of the Nessus server.)

Please note that the escorting firewall does not ever authenticate or authorize any user or system to perform an action. In this design, the escorting firewall is positioned behind a router or remote access server that is responsible for ensuring the authenticity and authorization of the traffic that it passes to the network. The escorting firewall merely attempts to detect policy violations (the aforementioned, configuration compliance and acceptable network usage policies) and respond by locking out the policy violators -- effectively revoking authorization. This is not a remote access solution; it is a remote access supplement.

This server performs operations based on four conditions:

1. When no users are connected
2. When users are connected
3. When a new user connects
4. When a user violates policy

Conditions 1 & 2:
When users are and are not connected the server will:

1. Scan all network traffic and check for policy violations

2. Wait for new users to connect
3. Bridge two network segments, allowing authorized traffic between them
4. Update the revoked user authorization list

Condition 3:
When new users connect the server will:

1. Verify that the user has not had its authorization to access the network revoked
2. Scan the system of the connecting user for policy violations

Condition 4:
When a user violates policy the server will:

1. Add the user to revoked user authorization list with the time of policy violation
2. Drop that user's traffic
3. Inform administration of the policy violation

Conditions All:
At no time will the server perform the following operations:

1. Bridge traffic for a user with revoked authorization
2. Permanently revoke a user's authorization to connect
3. Drop traffic for a user not in the revoked authorization list

In order to satisfy the specifications of the server, the server has been designed with the following components:

Hardware
Dell PowerEdge 600SC with 1.8 GHz Intel Celeron Processor
128MB RAM
40GB IDE HDD
1 additional 10/100 Intel Ethernet Controller (1 already on the motherboard)

This system assumes a low-traffic network. More CPU and RAM may be needed depending on use and security policy; less may work also. Depending on the configuration of the signature based scanning tools, disk performance usually limits the sustained maximum throughput before processors fail to keep up. However, testing showed that memory consumption attacks caused CPU utilization to reach levels so high that all traffic bridging stopped and even the local console became effectively unresponsive. Fortunately, to the best of the testing measurements ability to determine, the blocking functions continued to perform correctly up to the point when all network traffic stopped. Though risk mitigation will be covered in the next section, lightweight hardware is acceptable for fulfilling most of the specifications of this server. Faster and more robust memory, disk, and network configurations will increase the server's ability to withstand denial of service attacks, thus enabling it to fulfill its specified requirement to always bridge authorized traffic.

Software
OpenBSD version 3.5 (snapshot) http://www.openbsd.org
SNORT 2.0.0 (build 72) http://www.snort.org
NESSUS 2.0.10 for OpenBSD http://www.nessus.org
Custom scripts based on Snort2pf 3.2.

OpenBSD was chosen over other free operating systems for several reasons:

1. OpenBSD has a proven track record for security
2. OpenBSD provides kernel level support for the excellent pf firewall
3. OpenBSD has recently implemented several stack protection techniques to mitigate the risks of buffer overflows
4. OpenBSD has lower hardware requirements when compared to most popular Linux distributions
5. OpenBSD in this 3.5 release has implemented CARP, the Common Address Redundancy Protocol, which creates the potential for fault-tolerant escorting firewalls!
6. As opposed to other Unix-like operating systems it is completely free.

Snort was chosen because it is the definitive attack detection engine. It is licensed under the GPL and is therefore mostly free.

Nessus was chosen because it is a very robust vulnerability-scanning tool, is highly configurable, has a large user community, and also because it is GPL licensed. The Nessus client server design also adds the ability of the escorting firewall to be invisible to the systems it bridges as it can send scan requests out to a different system running the Nessus server on a third network interface. The design of the system in this step by step does not however rely on a second system running the Nessus daemon and as such, will not be invisible to the remote access users.

Snort2pf is a PERL script that monitors the snort alert file and dynamically adds block statements, using an anchor in the pf config file, for hosts that Snort detects are attacking the network. Snort2pf has an amnesty feature that removes the blocks after a configurable amount of time, default of three minutes. Snort2pf is really well written but lacks some important features that other similar tools have implemented. The first glaring feature omission is that it has no support for white lists -- lists of addresses or networks that should never get blocked. White lists are important because they prevent malicious users from launching denial of service attacks using network address spoofing. Additionally, as its name suggests, it only supports the pf firewall.

Before deploying the exact escorting firewall described here, look at using SnortSam, www.snortsam.net, as it features a client server design and an impressive list of supported firewalls.

Network Services
Inspects, filters, and passes traffic across interfaces in a bridging firewall capacity.
Syslog (514/udp)*
OpenSSH (22/tcp)*

*Access to these ports will be restricted to certain designated hosts. It is possible that neither of these services will be required in your environment. OpenSSH exists solely for remote administration and syslog is only used in this example to receive messages from the remote access server when new users connect. In your setup, you might choose to use SNMP or even pf to detect when new users connect. Don't limit your imagination to the technique that this guide uses. Though the focus so far has been on reducing the risks of allowing remote user access, technically the design could almost as easily be used to secure wireless access points, and publicly accessible Ethernet drops. From a technical point of view you are designing a simple automated system that enforces network security policies.

User Management
Other than the normal user accounts created for any administrators that are responsible for the management of the system, no other user accounts are needed. Technically Nessus requires that a user account be created but this is not an actual shell account; Nessus maintains its own user database. Additionally, as long as the Nessus daemon is running on the escorting firewall itself, then it is possible to bind it to the loopback interface – further reducing its significance.

Risk Mitigation Plan

A good risk mitigation plan is one that defines controls and procedures that are proportionate to the risks they mitigate. Just as you must weigh the costs of deploying and maintaining this server against the risk of leaving remote access users unchaperoned, you must also weigh the costs of each configuration decision against the risk that it reduces. Developing models to aid in practically and accurately determining risk in information technology environments is the subject of many dissertations. Regardless of the method you chose to analyze your risk, you will at some point have to identify what assets you have, the threats they face, the potential costs if those threats are realized, and the likelihood that they will be. It is this measurement that determines appropriate response. In the absence of sufficient data or human resources to accurately make this measurement, industry accepted standards and best practices should be used to determine response. The following risk mitigation plan is based on both best practices and generalizations about the threats, costs, and likelihood of certain attacks.

Fortunately, the net cost of risks from partial compromises, including avoiding detection of system configuration compliance and acceptable network

usage policy violations, are only the investment and opportunity costs of deploying and maintaining this system less the positive value added with each successful policy violation detection and response. Ideally, the costs of deploying and maintaining this system will be so low that if the system only stops one malicious attack or prevents one destructive worm outbreak on your trusted network, it will have paid for itself. However, depending on your configuration, in the worst-case scenario, total compromise of this machine allows an attacker to more easily find vulnerabilities in other systems, disrupt remote user access, monitor the traffic of other users, and distract security administrators from other tasks. Depending on your assessment of the costs and likelihood of total system compromise, these additional risks introduced by this system may or may not be a significant concern for your organization.

What are the sources of threats? The escorting firewall has only two main sources of threats, malicious attackers and Internet worms. Internet worms are a special case threat source and will be discussed later. Because the purpose of the escorting firewall is to prevent attackers from accessing the trusted network, malicious users already on the trusted network will be ignored as sources of threats. Though malicious users on the trusted network may very well attack the system out of malice, ignorance, or to abet an outside attacker, their mere existence implies a partial compromise.

Your escorting firewall implicitly already has some level of protection because it is positioned behind your remote access server that should only be allowing network access to presumably trustworthy authorized and authenticated users. From this you can already infer quite a lot about your attacker. First and obviously, they either fully compromised the remote access server or they already have at least one set of valid credentials, legitimately or not. Second, your attacker is probably highly motivated. Third, there is a good chance that they have probed other systems and people and know a lot about the design of the network and the location of their ultimate targets. In summary, you face a low probability of a highly motivated, well-informed, well-equipped attacker reaching the escorting firewall with goals of compromising one or more high-value targets on the trusted network; plan accordingly.

Skilled attackers are excited by security systems that implement automated response. The attackers like them because they create an opportunity to be used against the systems they are designed to protect. In addition to the inherent risks associated with automated response, sensitive information might be stored in logs and configurations that could aid the attacker in attempts to compromise other network resources. The keys to mitigating the risks associated with a compromise of this system are to reduce its threat surface with system hardening and to minimize its known vulnerabilities with aggressive patch management. These two keys will be repeatedly referenced throughout this paper.

For a weak force to succeed in asymmetric warfare it must find ways to use the stronger opponents advantages against them; your opponent will try to use this system against you. However, even accounting for its inherent attractiveness, your escorting firewall will be a high-cost, medium-low reward

target for an attacker. If an attacker fails to find an easily exploitable vulnerability quickly it will try to go around this system rather than through it. Nevertheless, it is necessary to assess the specific threats posed by full and partial compromises of your escorting firewall that is not only an automated response system but also a critical network link, bridging remote access users to your trusted network.

The obvious threat is from an attacker using the system in a denial of service attack by tricking the system into restricting access to legitimate network resources. A common example of this is an attacker spoofing attack packets originating from DNS servers. A poorly designed or configured automated response system might incorrectly block access to the DNS servers likely having the effect of bringing everything from E-mail and web surfing to database and network file access to a halt. Depending on your setup, printing might even be dependent on DNS. And this is supposed to reduce my risk? Fortunately you'll configure the aforementioned "white lists" that inform the escorting firewall which hosts are never allowed to be blocked.

Another important threat stems from the Nessus vulnerability scanner. Ironically, Nessus is one of the many security tools that are ideally suited for both attackers and defenders. Attackers will attempt to use your Nessus scanner to assist their selection of subsequent targets on your network. As security auditors and administrators, you must use it to find vulnerable network systems that need software updates. Whether the existence of these tools has ultimately helped or hurt security overall, the fact they do exist dictates that you use them. It is essential that you know your network and your systems better than the attackers targeting them. You will mitigate the risk of attackers using your Nessus daemon against you in three ways.

First, restrict network access to the Nessus daemon itself. If you are running the daemon on the same system as the client, bind the daemon to the loopback adaptor. If you choose to run the daemon on a different system, use both the Nessus rules file and a host-based firewall to restrict access to only those systems from which authorized Nessus users are authorized to initiate scans. Also, depending on your security policy you might wish change the default TCP port Nessus listens on to something other than the default. There are good management based arguments for leaving daemons on their default ports, and they are usually augmented with a clichéd statement, "no security through obscurity." Still, the argument for it is the other buzzword security strategy, defense-in-depth.

Secondly, limit the abuse potential of Nessus. Nessus allows you to specify in its rules file, which target hosts and networks to permit and which to deny. Additionally, require strong passwords for your Nessus users – remember, it maintains a user account database completely independent of the local shell accounts that exist on the server. You should already have security policies that dictate correct password usage, follow them.

Thirdly, remember to protect the result files that Nessus generates. If you use the tool regularly but are sloppy about handling the results files, your attacker won't even need to make any noise by running the scans because you will have already done it for her. Also, remember that if you create cron jobs that run scans

automatically and E-mail you the results, make sure those messages aren't sent in cleartext on the wire.

Nessus introduces two other risks in addition to its potential for abuse, compromise and denial of service. Though the virtues of aggressive patch management are stressed later, it is especially critical for both Nessus and Snort. Each time an attacker connects with an unblocked user account he will automatically get scanned by Nessus. If the Nessus engine is found to expose an unchecked buffer or other vulnerability that can lead to compromise by the hosts it scans, an equipped attacker will have root. Though this author has not tested it, it may be possible for an attacker to create a denial of service attack against the escorting firewall and potentially access for all remote users by using a tool such as LaBrea to tie up the scanner. While this would without doubt prevent a successful scan for configuration compliance, it would only tie up one process as your scripts will spawn a separate client for each connecting host and the daemon is designed to fork a new process for each scan request as well. Also remember that Snort is still looking for attacks and OpenBSD has a very robust TCP/IP stack that was designed to handle abuse well.

Avoiding detection is a high priority for the attacker. Because of this there is a risk from the potential that an attacker might attempt to gain access to Snort. Careful attackers will first attempt to identify your ruleset such that they can avoid attacks that they know will be identified. If their attack requires actions that will be identified, their next step will be to manipulate Snort itself by either modifying its rules file to ignore their actions or by modifying its alert file to hide their activities. It is not likely that the attacker will outright disable Snort as doing so would draw the attention of the log monitors. Of course if they can tell from profiling network traffic that no one actually checks the logs, you have probably already lost.

Other than the remarkable protections afforded by basic OpenBSD system hardening and patch management, this practical will not detail ways to mitigate the risks of log manipulation. Strategies you might wish to use however include regularly refreshing the Snort rules database, backing up your log files to a remote system, running a secure file monitoring application such as tripwire, and injecting artificial alerts in an attempt to catch an attacker indiscriminately deleting log entries using ranges of time. Implementing tripwire is a good idea for one other important reason -- change management. It is very difficult to maintain good security without being organized.

An important organizational responsibility that security staff should impress upon the network and server administrators is that they keep detailed logs explaining what, when, and why changes are made. Please note that Tripwire is not a change management system. Tripwire provides the ability to efficiently and effectively audit your change management system whether it is trouble-ticket based or version control oriented, simply by recording changes to the files you configure it to watch. Again, you should already have written security policies and procedures that define change management for your organization. While, Tripwire isn't configured in this practical, using OpenBSD's /etc/changelist file and /etc/security script will provide a rudimentary form of this functionality. In many cases, the OpenBSD system is more than sufficient for this task. However,

the OpenBSD system lacks most of the security features that make the commercial Tripwire system the de facto standard tool for this purpose.

As stated earlier, this risk mitigation plan relies on two basic strategies: reducing the server's threat surface and minimizing known vulnerabilities with aggressive patch management. Keeping Snort, Nessus, and OpenBSD patched are critical to preventing system compromise. On Friday March 19, 2004 at 8:45 pm PST a destructive worm named Witty, efficiently infected a relatively small population of Internet connected systems running vulnerable firewalls just one day after the vulnerability announcement. Almost all of the vulnerable systems had been compromised within 45 minutes. This is a great time to reassess your vulnerability management strategy and your patch management procedures.

Before Witty it was reasonable to wait a week or longer to patch moderately obscure Internet connected daemons with publicly documented vulnerabilities. A week gave enough time for patch testing and an opportunity to bring down the servers during regularly scheduled maintenance windows. At this point, patches lead to more vulnerability related downtime than attackers' exploits. Usually however, problems relating to bad patches can be resolved relatively quickly or rolled back. Best practices dictate that all incidents involving remote vulnerability exploitation require an investigation, rebuild, and restore of the affected system. In many cases, if it is not detected quickly, it might not even be possible to determine the original vulnerability that allowed the intrusion or to accurately assess exactly which assets were compromised. Again, risk is unavoidable. However, if you are the person held accountable when a major incident occurs, remember that after Witty – while it may not be cost advantageous – it might be considered reasonable to deploy patches for critical and exposed systems within a time frame as short as two to eight hours. Do yourself a favor and remind your management in writing of the costs and risks associated with your patch management policies and procedures.

One of the reasons for implementing an escorting firewall is to prevent the systems of authorized remote access users from spreading Internet worms to the systems behind your firewall on your trusted network. Keeping your Snort rules current up to the minute, possibly by subscribing to the Snort rules mailing, is an essential part of keeping you escorting firewall significant. The other escorting firewall configuration to limit the spread of Internet worms to your trusted network is pf. Just because you let your remote access users have an IP address on your LAN does not mean that you have to give them all of the layer four through seven opportunities that come with it. Identify what kinds of access the remote users will need, permit it, but then use a default deny to prevent all other types of traffic. Would you not hate it if the next Internet worm, based on a vulnerability in Microsoft's Universal Plug and Play port, made it to your network before you had the Snort rule for it?

Also, remember the significance of that hardware. It is amazing that a system that does this much can work so well 99.99% of the time on junk hardware too slow to run turn of the millennia version of Microsoft Office. However, when a determined attacker finds your escorting firewall in the path to its target, you are now dealing with the .01% of the time. Start off with the

hardware available, but remember that more network interfaces, RAM, and faster disks will be needed when the expert cracker finds you.

Steps to Install and Harden the Server

Installing OpenBSD can be very simple; it can also be very difficult. OpenBSD has a very good FAQ and its man pages have a very strong reputation. One thing that OpenBSD does not have, that can be quite crippling, is a publicly active user community filled with zealots that are willing to do whatever is needed to make it work for you. In fact, the OpenBSD community's reputation for brash arrogance and a general lack of concern with whether anyone uses OpenBSD or not, far exceeds the reputation of its thorough man pages. If you try to install OpenBSD and have hardware problems, or software you like has issues compiling or running, finding help may be difficult and while others will not -- this author probably cannot, help you. OpenBSD was not selected for its ease of installation but rather for its security and ease of management once the wrinkles of setup have been conquered.

Setting up your escorting firewall server -- summary of steps:

Step 0: Assess your network, site policies, procedures, and government laws. Compile all relevant information and have it available; many of the configurations made in steps 1 through 8 will require customization for your particular environment. Running Snort monitors and logs network traffic, it might not be legal to do so where you live, or it may require notifying your users. Nessus scans will likely register as attacks on the systems of the users you are targeting, this may be illegal and if improperly configured are quite likely to crash the systems you are scanning. Super DMCA states such as Illinois, Florida, Arkansas, Delaware, and Pennsylvania (http://www.eff.org/IP/DMCA/states/#affectedstates) have laws that probably make some of these actions illegal for anyone living in them. Getting legal advice before setting up this system is a terrific idea.

Step 1: Creating bootable CD-ROMs for installation.
Step 2: Configuring the installation environment, filesystem, base packages, and installing.
Step 3: Post install tasks, including user setup, basic hardening, and patch installation.
Step 4: Configuring the system for the network including configuring pf.
Step 5: Installing Snort, Nessus, and PERL scripts.
Step 6: Configuring Snort.
Step 7: Configuring Nessus and the PERL scripts.
Step 8: Verifying the correctness of the design and auditing the system.

Step 1

Unlike many other popular UNIX-like operating systems, OpenBSD does not provide CD-ROM ISO image download files. Instead, if you are in the US, for $40 USD plus shipping from Canada, you may purchase a CD-ROM set and have it mailed to you, or alternatively you can download the files for your system's architecture and possibly create your own bootable CD-ROM.

If you download the files, be sure to check the MD5 sums before burning your CDs. Best practices dictate that you get the MD5 sums from a site other than the one you downloaded your files from. By the time you read this, after May 1, 2004, OpenBSD 3.5 will have been released.

The OpenBSD project releases new versions every six months. Because of the proximity to the 3.5 release date this guide has been written using multiple development snapshots of OpenBSD 3.5. As there have been no changes during the last month that have impacted the steps outlined in this guide, hopefully nothing will change in the final release that prevents successful installation using only these steps. If you want to support great security and you haven't used OpenBSD much in the past, you should consider buying a CD-ROM set as they are quite useful. If you have used OpenBSD in the past then you probably already have a ton their tee shirts, posters, and stickers and can manage to put together your own CD sets easily. Here is what the OpenBSD website has to say about its CD-ROMs:

- Three CDROMs in a jewel case.
- The complete install components for **SIX** architectures: **i386**, **vax**, **amd64**, **macppc**, **sparc**, **sparc64**.
- The following architectures only available via FTP download: **alpha**, **hppa**, **hp300**, **mvme68k**, **mvme88k**, **mac68k**, **cats**.
- The CDs are bootable on i386, amd64, macppc, sparc, and sparc64.
- A funky and surprisingly artistic CD insert sheet which contains installation instructions. The information on this piece of paper makes OpenBSD somewhat easier to install than if you do an FTP install.
- A full source tree (ready for AnonCVS use).
- The latest reliable **XFree86** binaries for all architectures
- The latest **XFree86** source code with small modifications by us to make it prettier and more secure.
- Our own ports tree which has improved an insane amount since OpenBSD 3.4. Almost all packages work on almost all architectures.
- Several pre-built binary packages for the most common architectures, which are very easy to install directly off the CDROM.
- As always, stickers included!
- And many other things...

Creating bootable CD-ROMs for the i386 platform isn't terribly difficult if you have the following tools: a fast Internet connection, 600MB of free disk space, a device capable of writing CD-ROMs, and a program such as Ahead's Nero that is capable of creating bootable CD images using 2.88 MB floppy-disk emulation. A list of methods and mirrors for downloading OpenBSD is available at: http://www.openbsd.org/ftp.html. Using whichever method is most convenient for you get at least your platform's base, packages and tools directories. If you are in the US near Boulder, Colorado and have an i386 machine you might want to do the following:

Create a directory called 3.5 on your computer.

In it, download the i386 directory found at: ftp://ftp3.usa.openbsd.org/pub/OpenBSD/3.5/

In the local 3.5 directory create an additional directory called packages.

In it, download the i386 directory found at: ftp://ftp3.usa.openbsd.org/pub/OpenBSD/3.5/packages/

Back in the local 3.5 directory, download the tools directory found at: ftp://ftp3.usa.openbsd.org/pub/OpenBSD/3.5/

And if you are a fan of the ports system, you'll want to get the ports tree, found at: ftp://ftp3.usa.openbsd.org/pub/OpenBSD/3.5/ports.tar.gz

Once the above are copied, download the MD5 sums from at least one additional mirror and the main OpenBSD site itself. After ensuring the MD5 sums you are using match the md5 sums on the other servers, compute your own md5 sums and compare with the servers. As long as everything matches it is reasonable to assume that the files have not been altered since they were posted. If anything doesn't match, investigate and pending nothing was corrupted during transfer, inform Theo de Raadt of the problem: deraadt@openbsd.org.

If you use Ahead's Nero software to write CD-ROMs, start a "new compilation," selecting "CD-ROM (Boot)" to set the defaults. Under the boot tab of the compilation configuration screen, select image file. The image file you want to use is in the 3.5/i386 directory that you just downloaded and is named cdrom35.fs, be sure to select "2.88MB Floppy Emulation." To prevent compatibility and installation problems, try not to use any features that are not strictly supported by the basic ISO9660 standard. Add the 3.5 directory to new compilation but remove the file bsd.rd from the 3.5/i386 directory. The bsd.rd file is the kernel for embedded systems and failing to remove it from the CD-ROM will make selecting your base sets very annoying. At this point cross your fingers and start writing the CD.

Step 2

With the system disconnected from the network, boot off of your new OpenBSD CD-ROM. The first installation task that will have a bearing on security is the configuration of OpenBSD partitions. OpenBSD's featureless fdisk utility is not one of the operating system's greatest strengths. Before continuing, you need to ensure that you are familiar with OpenBSD's partitioning lexicon and peculiarities. From the OpenBSD FAQ at http://www.openbsd.org/faq/faq4.html#Disks:

> 4.5.2 - Setting up disks
>
> Setting up disks in OpenBSD varies a bit between platforms. For i386 and macppc, disk setup is done in two stages. First, the OpenBSD slice of the hard disk is defined using fdisk(8), then that slice is subdivided into OpenBSD partitions using disklabel(8).
>
> Some users may be a little confused by the terminology used here. It will appear we are using the word "partition" in two different ways. This observation is correct. There are two layers of partitioning in several OpenBSD platforms, the first, one could consider the Operating System partitioning, which is how multiple OSs on one computer mark out their own space on the disk, and the second one is how the OpenBSD partition is sub-partitioned into individual filesystems. The first layer is visible as a disk partition to DOS, Windows, and any other OS that can coexist with other Operating Systems on the IBM AT descended machines. The second layer of partitioning is visible only to OpenBSD and those OSs which can directly read an OpenBSD filesystem.

In OpenBSD's defense, the term partition when applied to computer hard disks has no clear specific meaning; just look at Google's definition search for the word partition: http://www.google.com/search?q=define:partition. From this point forward, 'slice' refers to the A6 OpenBSD primary partition and 'partition' refers to any sub-partition of an OpenBSD slice that are used to store its filesystems and swap space.

The Dell Poweredge servers come with a handy maintenance partition that you might not want to delete. Unfortunately, OpenBSD's fdisk utility is the most user-unfriendly disk preparation tool this author has ever used. The easiest situation is when you can give the entire disk to OpenBSD. If, as in this case, you have a Dell utility partition you want to keep, hopefully this fdisk output will help.

```
# /sbin/fdisk wd0
Disk: wd0       geometry: 4863/255/63 [78124095 Sectors]
Offset: 0       Signature: 0xAA55
         Starting       Ending       LBA Info:
 #: id    C   H  S -    C   H  S [       start:      size   ]
------------------------------------------------------------------------
 0: DE    0   1  1 -    6 254 63 [          63:    112392 ] Dell Maint
*1: A6    7   0  1 - 4862 254 63 [      112455:  78011640 ] OpenBSD
 2: 00    0   0  0 -    0   0  0 [           0:         0 ] unused
 3: 00    0   0  0 -    0   0  0 [           0:         0 ] unused
```

There really should not be any reason why you would want to load other operating systems on a production escorting firewall. If however you just want to mess around, these sites may help you with the trickiness that is multi-booting OpenBSD and anything else:

- http://www.gainesville2600.org/stuff/obsd_linux_win2k_howto.htm
- http://geodsoft.com/howto/dualboot/
- http://www.openbsd.org/faq/faq4.html#Disks

The installer recommends that you at least create separate partitions for /, swap, /tmp, /var, /usr, and /home. Later you might want to mount both / and /usr read-only in an attempt to defeat malicious code and the lesser skilled attackers that rely on it. Nodev, noexec and nosuid mount options will also be applied to certain partitions. The trickiest partition is /var as it will store logs and backups and will certainly be targeted for attack.

Giving /var its own partition will allow your system to be more fault-tolerant against log flooding via Snort and syslog as well as other disk consumption attacks. However, given the role of this system it might be preferable to have the system halt because the automated blocking functionality of this system is dependent on reading new messages out of the log files on /var. The distinction here is between failing open and failing closed. If this were a network database or file server you would not want a crash to occur that could corrupt important data, you would want the system to fail open for this kind of attack. The escorting firewall server's primary purpose is to enforce network policy, which it can no longer do effectively when /var goes down. Nevertheless, this author recommends creating a separate partition for /var and implementing procedures for monitoring its disk usage.

Format all partitions using the OpenBSD filesystem except swap, which performs better without the overhead of a filesystem. The following are partition

recommendations for servers, such as the escorting firewall, that that do not need to provide much space for normal users:

Device Name	Mount Point	Minimum Size	Max needed	Special Mount Options	Role
wd0a	/	60MB	1GB	-ur (Read-Only)	Stores /, /bin, /boot, /bsd, /dev, /etc, /mnt, /root, /sbin, /stand, /sys
wd0b	swap	Equal to RAM	Twice RAM		Serves as slower primary storage
wd0d	/tmp	120MB	35% if used for full backups	-uw -o nodev, nosuid, noexec	Stores temporary files of some programs including ports compiler and also for backups
wd0e	/usr	500MB	10%	-ur -o nodev	Stores applications and scripts
wd0f	/home	100MB	2GB	-uw -o nodev, nosuid	Individual profile and file storage
wd0g	/var	4GB	Remaining or 50%	-uw -o nodev, nosuid, noexec	Stores logs and backups

```
# disklabel /dev/wd0c
# using MBR partition 1: type A6 off 112455 (0x1b747) size 78011640 (0x4a65cf8)
# /dev/wd0c:
type: ESDI
disk: ESDI/IDE disk
label: IC35L040AVVA07-0
flags:
bytes/sector: 512
sectors/track: 63
tracks/cylinder: 16
sectors/cylinder: 1008
cylinders: 16383
total sectors: 78125000
rpm: 3600
interleave: 1
trackskew: 0
cylinderskew: 0
headswitch: 0           # microseconds
track-to-track seek: 0  # microseconds
drivedata: 0

16 partitions:
#        size   offset    fstype   [fsize bsize   cpg]
  a:   163737   112455    4.2BSD     2048 16384   162   # (Cyl.  111*- 273)
  b:   614880   276192      swap                        # (Cyl.  274 - 883)
  c: 78125000        0    unused        0     0         # (Cyl.    0 - 77504*)
  d:  1228752   891072    4.2BSD     2048 16384   328   # (Cyl.  884 - 2102)
  e:  6291936  2119824    4.2BSD     2048 16384   328   # (Cyl. 2103 - 8344)
  g:  8388576  8411760    4.2BSD     2048 16384   328   # (Cyl. 8345 - 16666)
  h: 61323759 16800336    4.2BSD     2048 16384   328   # (Cyl. 16667 - 77504*)
  i:   112392        63   unknown                       # (Cyl.    0*- 111*)
```

Do not try to setup the special mount options until after the install is complete and the system reaches a stable state. Once the file systems have been formatted you'll be asked for a hostname and then asked if you wish to

configure the network. Do not configure the network at this time. Next you will need to create a password for the root account. Follow your organization's password policy and attempt to exceed its minimum requirements. If you do not already have a password policy, pick a password that is at minimum eight characters and contains mixed case, punctuation, and numerals. If you prefer something easier to remember but more difficult to type, choose an easy to remember passphrase that exceeds twenty characters but may be all lowercase and not contain any punctuation or numerals. The latter is actually more secure for three reasons: 1) 95^8 is way less than 26^20, 2) easier to remember passwords are less likely to be written down and lost, 3) it is less likely to be reused to protect other systems because most systems will not support 20 character long passwords.

The password space estimates assume that the attacker knows which types of characters are and are not being used but believes them to be random. The 95 potential characters per position is derived from ten numerals, 26 lowercase letters, 26 uppercase letters, and thirty-three math and punctuation characters. Extended ASCII characters are not included. Obviously a dictionary could be created to attack common phrases or only use English words but this author has yet to find any. In the interest of full disclosure a two-fish generated twenty-character password was included, it is clearly impossible to memorize.

jTz^qAjK	= 95^8 =	6634204312890625 (generated by two-fish)
maryhadalittlelamb	= 26^18 =	2947951020001391888644085 76
everywherethatmarywent	= 26^20 =	19928148895209409152340197376
eaxuqqpypdduuoysdlut	= 26^20 =	19928148895209409152340197376

When selecting the install sets, use the defaults minus game35.tgz and misc35.tgz. No one needs to play games on the escorting firewall server and no one should need X to read logs and edit PERL scripts. Following that, the installer will ask if you wish to start OpenSSH by default, to which you will answer yes. Answer the X question with no, unless you really need it. Select your local time zone and you are done.

Step 3

In step 3 you're going to handle a number of post-installation tasks that include system hardening, configuring the system for users, and installing patches. Every single configuration file that you modify should be added to /etc/changelist if it isn't already there. /etc/security creates a backup of every file listed in /etc/changelist each day as scheduled by cron (actually /etc/daily is scheduled by cron and it starts /etc/security).

Restart and login as root. Use the /usr/sbin/adduser command to setup adduser.conf and then create your users. You will like OpenBSD a lot more if you use ksh for your default shell. Setting it as the default shell for all new users in adduser.conf will win favor with other administrators. Using default for the default login class is safe. If you are only going to allow administrators to have shell accounts as suggested, you may want to set the default to staff as it imposes fewer restrictions, look at the sample /etc/login.conf file and man pages for more

details. If you choose to copy the dot files (.cshrc, .login, .mailrc, .profile, .rhosts) from /etc/skel by default, make sure to look at each one and ensure that they are configured to your satisfaction. Since you'll be editing PERL scripts later you may want to install vim which will colorize the scripts if TERM=xterm-color is in the .profile. Definitely use blowfish for password encryption.

If you want to change root's shell to ksh, do that now with the command /usr/bin/chsh. After you've created the needed normal user accounts for your administrators, and placed them in wheel, logoff. Log back in as the normal user account you created for yourself and su to root. As soon as you added the first normal user account to the wheel group, all other normal accounts, not in group wheel, lost their ability to su. If administration has been delegated more granularly than the all or nothing wheel group, now is the time to setup the sudoers file using /usr/sbin/visudo. It is always a good idea to uncomment the line "# %wheel ALL=(ALL) ALL". Uncommented, that line allows users in the wheel group to run commands as root without actually starting a new shell as root. That is useful because of the temptation to stay logged in with su all of the time, defeating the purpose of creating normal user accounts.

It should go without saying that unrestricted physical access to this system will result in a full compromise. There may be situations however where even your most secured office space allows limited access to persons not authorized to operate the server. No matter what steps you take, if an attacker gets access to the data on the hard drive, you will eventually lose. With this in mind you should secure your system in such a way that all possible methods of accessing the data on the hard drive take at least as long as removing the hard drive and copying its contents to another system. Limiting boot devices to the hard drive, physically removing unneeded removable storage devices, password protecting the bios, and using a case lock will help. The more an attacker knows about the configuration of the system prior to their physical access attack, the less likely you will be able to detect it. Again, if an insider is assisting in the attack, there has already been at minimum a partial compromise.

If an attacker is only able to achieve restricted physical access, meaning they aren't able to open the case or restart the system, it might be valuable to remove root's ability to log in locally, or at least restrict root to an obscure terminal such as ttyc5 (ctrl-alt-6). You may do this by editing the /etc/ttys file and removing the secure at the end of each of the ttyCn lines. If you want to force a root password to be entered for single user mdoe, add secure to the end of the console line. You might also want to consider restricting local access to only the serial port.

Next you will want to take some basic network hardening steps prior to configuring the network interfaces. These steps will include locking down sshd, disabling unneeded services, and loading up some basic pf rules.

Secure Shell (SSH) is a critical tool for securely administering computers. SSH is an extremely powerful replacement for insecure protocols and services such as telnet, ftp, and many of the "r" utilities. There are a number of things to

consider when configuring the /etc/ssh/sshd_config file. The first and easiest change to make is to uncomment the line "# PermitRootLogin yes" and change its value to no. Also adjust the way OpenSSH handles attackers attempting to deny service to the sshd daemon. With the default configuration an attacker only needs to open 10 connections to the sshd daemon to prevent any other users from accessing the daemon. OpenSSH will allow each connection to remain open without a login attempt for two minutes. This means that it is trivial for an attacker to prevent an administrator from gaining access to the system.

There are two values in the sshd_config that you can use to increase the difficulty of tying up sshd for an attacker. First, uncomment and decrease the value of "#LoginGraceTime 2m" to thirty seconds, "LoginGraceTime 30". This gives you more chances to beat the attacker by forcing his connections to timeout faster. Second adjust the value of "# MaxStartups 10" to either something much higher or alternatively enable early random drops. Configuring early random drops instructs the sshd server to start dropping a percentage of new connections after a certain threshold of startups is reached until another higher specified threshold is reached whereby all new connections are dropped. The line "MaxStartups 4:70:20" instructs sshd to drop 70% of new connections after four unauthenticated connections are established and once there are twenty unauthenticated connections, drop all new connection attempts.

For reference, each unauthenticated connection requires a new instance of sshd that itself consumes approximately 312KB of memory (text, data, and stack) and about 1108KB of resident memory. It follows that 20 unauthenticated connections tie up over 22MB of resident memory. In the case of this escorting firewall with 128MB of memory running Snort, Nessusd, and the PERL scripts resulted in a peak memory consumption of about 90MB during multiple Nessus scans with one interactive user. To prevent an SSH attack from tying up too much memory and slowing down Snort or the Nessus scans, the value "4:70:20" was selected for MaxStartups on this Dell.

Another sshd configuration options to consider disabling is ssh protocol 1. This can be accomplished by changing the line "#Protocol 2,1" to "Protocol 2". This author feels that SSH protocol 1 as implemented in OpenSSH 3.8.1 is secure and may be left enabled. That said, most security experts recommend disabling it, and protocol version 2 is cleaner and more structured anyway. In the risk mitigation plan, the outlined strategy defined two goals: reducing the threat surface and eliminating known vulnerabilities. Disabling protocol 1 reduces the threat surface with minimal cost so it should be done. If your server configuration policy requires that you change the default port of daemons when possible then you can make the sshd daemon listen on a more obscure port by uncommenting and changing the line "#Port 22".

If you only need access to the sshd daemon from an internal or administrative network, then in addition to restricting access with pf, you can force sshd to listen only on your internal or administration network IP address(-es). To do this, uncomment and change the line "#ListenAddress 0.0.0.0". Read the sshd_config man page to avoid making mistakes with the port configuration. A feature of ssh that's awesomeness is surpassed only by its scariness is TCP

forwarding. TCP forwarding allows clients to proxy TCP traffic through the sshd daemon. There are an infinite number of cool uses for this feature; read the many practicals on setting up bastion hosts with SSH. In this example, disabling tcp forwarding does not improve security significantly because anyone that can login to the escorting firewall through sshd can probably already setup forwarding with /usr/bin/nc or some other utility. If you want to turn it off uncomment and change the line "#AllowTcpForwarding yes" to "AllowTcpForwarding no". Verbose logging can be enabled and might be useful. The default logging parameters are "#SyslogFacility AUTH" and "#LogLevel INFO". If your site policy requires login banners, sshd has them, "Banner /path/to/file".

SSH protocol 2 gives you more control over the encryption and hashing algorithms used to protect the integrity and confidentiality of the connection. If laws or your site policies dictate specific algorithms or key sizes, use the Cipher and MACs commands. The defaults are not included in the sshd_config file by default but if they were the would look like:

```
#Cipher aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,arcfour,aes192-cbc,aes256-cbc,aes128-ctr,aes192-ctr,aes256-ctr
#MACs  hmac-md5,hmac-sha1,hmac-ripemd160,hmac-sha1-96,hmac-md5-96
```

Read the sshd_config man page. Every setting might be important depending on your environment. If you have setup Kerberos, RADIUS, a PKI, or some other central authentication authority on your network you will want to configure sshd to use it. Also read the sshd man page. Some of the sshd flags are not easily configurable in the sshd_config file. To enable them, add them in the /etc/rc.conf.local file on the line "sshd_flags="". Setting sshd_flags="-4" forces IP version 4 addresses only.

Other network changes to make are in the /etc/inetd.conf file. If you do not need the ident, daytime, or echo ports remove them by commenting out their respective tcp and tcp6 lines. In the past, rstatd and rusersd were enabled by default. You will notice that they are now commented out by default. The portmap value in /etc/rc.conf was set to no as well. These values control rpc, which is used to provide access to NFS and yp. To be really thorough you will want to comment out the portmap lines in /etc/rc as well. As you go through these files you will notice that sendmail is configured to run as a daemon by default. Here is what George Shaffer, author of the excellent but out of dated geodsoft.com OpenBSD hardening guide (http://geodsoft.comhowto/harden), wrote about it:

> "Prior to OpenBSD 3.0, I always disabled sendmail as a daemon by changing the sendmail_flags to NO in /etc/rc.conf and also commenting out the /etc/rc lines that started sendmail. Now there is a change that requires sendmail to run as a daemon. I find this odd and annoying but it seems to be fact and apparently a Sendmail version issue, not an OpenBSD issue. There were serveral references to this in misc@openbsd.org."

This author leaves sendmail enabled because it leaves intact the primary method that the /etc/daily script delivers its reports to the administrators. The security benefit outweighs the additional risk. A "/usr/bin/netstat -na | /usr/bin/grep "\.25"" reveals that sendmail is only listening on 127.0.0.1 and its IPv6 equivalent by default. However, a more careful observation of the output from "/usr/bin/netstat –na" reveals that a daemon is listening on tcp 587, the RFC

2476 MTA submission port, and udp 512, the comsat/biff mail notification port. These are also related to sendmail, but it again is only listening on the IPv4 and v6 loopback addresses. If you see any "*." lines other than "*.22" in the "/usr/bin/netstat –na" output, investigate. At this point, you are almost ready to configure the network.

On a separate computer, check the OpenBSD website for security alerts, http://www.openbsd.org/security.html#35. As of now there are no vulnerabilities for 3.5, if there were, you would download the patches, verify the md5sums, and install them on your escorting firewall server.

Step 4

In this step you will configure the system for the network. This step includes setting up bridging, configuring your interfaces, and configuring pf.

First, restart the server and verify that any previous changes you made were effective. Next, login and su to root. Enter the "/sbin/ifconfig –a" command to see what network interfaces OpenBSD has detected and installed. To configure an interface create a file in /etc called hostname.INTERFACE where INTERFACE is the name of an interface that shows up in ifconfig. For this server the following was configured:

```
# /bin/cat /etc/hostname.em0
up
# /bin/cat /etc/hostname.fxp0
inet 192.168.6.22 255.255.255.0 NONE
#
```

If you have an administration network interface you would configure that interface as well. Notice that fxp0 was configured with an IP address. This is the "outside" interface. In this example the outside interface needs an IP address to receive syslog messages from remote access device, the PIX firewall. Also note that the "outside" interface is actually on the same subnet as the internal network, so it is only outside relative to the other interface. Since the Nessus daemon is going to be configured on this machine as well it also needs an address from which Nessus scans will be sent. Hopefully, you will be able to have a separate administrative network interface and a Nessus server running on a different system such that the escorting firewall can run in a stealthier configuration, thus further reducing its threat surface.

To configure bridging you need to make at most three changes. First uncomment the line "# net.inet.ip.forwarding=1" in the file /etc/sysctl.conf. This should be the only uncommented line in the file. Second, create the file /etc/bridgename.bridge0. In the file write a line adding the interfaces you want to bridge. See the example below. Third, if any of the interfaces listed in the bridge file haven't yet been configured create their respective /etc/hostname.inteface file and if no IP address is needed, merely insert the line "up".

```
# /bin/cat /etc/bridgename.bridge0
add fxp0 add em0 up
```

At this point, the bridge is configured and before you restart you should write your pf rules and enable pf. To enable pf to start on reboot, edit the file /etc/rc.conf and set pf=yes. Next edit the /etc/pf.conf file. If you are not familiar with pf, read the faq: http://www.openbsd.org/faq/pf/ and the man pages for pf.conf, pf, and pfctl. Alternatively you can look at the config file configured for this example, as it is fairly intuitive.

```
# /bin/cat /etc/pf.conf
#       $OpenBSD: pf.conf,v 1.27 2004/03/02 20:13:55 cedric Exp $
#
# See pf.conf(5) and /usr/share/pf for syntax and examples.

# Interface aliases should be created for ease of administration.

outside_if="fxp0"
inside_if="em0"

#"Scrubbing" is the normalization of packets so there are no ambiguities in interpretation by the ultimate
# destination of the packet. The scrub directive also reassembles fragmented packets, protecting some operating
# systems from some forms of attack, and drops TCP packets that have invalid flag combinations.

scrub in all

#For traffic coming from the trusted network, allow all.
#      pf rules are best match and exit not first match and exit. The quick parameter forces first match behavior.
pass in quick on $inside_if all
#Each packet traversing the pf firewall is actually processed twice. First, the traffic comes 'in' on an interface.
#      Next, the traffic is processed again going 'out' on a different interface.
#Allow all traffic to leave out on the inside interface.
pass out quick on $inside_if all


# Block and LOG everything coming in and going out on the outside interface by default
#      Note that we specified both drop and log but not quick.
#      The drop command instructs pf to silently drop matching requests not reset them.
#      The log command tells pf to log the dropped packet.
#      Quick isn't specified because we don't want to filter everything, only the packets, that don't have better
#        matches.
block drop in log on $outside_if all
block drop out log on $outside_if all

########
# This section allows traffic to traverse from the least trusted interface (outside) to the other interfaces (inside)
########

# Allow ssh
pass in quick on $outside_if proto tcp from any to any port = ssh keep state

# Allow NTP and syslog
pass in quick on $outside_if proto udp from any to any port = ntp keep state
#192.168.6.1 is the remote access server and 192.168.6.69 is the RADIUS server, both need to send syslog to the
#      escorting firewall
pass in quick on $outside_if proto udp from {192.168.6.1, 192.168.6.69} to 192.168.6.22 port = 514 keep state

# Allow ICMP (echo request)
pass in quick on $outside_if inet proto icmp all icmp-type 8 code 0 keep state

########
# This section allows traffic to traverse from the trusted (inside) interface to the other (outside) interfaces
########
```

```
# Allow ICMP (echo-request)
pass out on $outside_if inet proto icmp all icmp-type 8 code 0 keep state

# Allow all UDP/TCP OUT and keep state
pass out on $outside_if proto udp all keep state
pass out on $outside_if proto tcp all modulate state

#This anchor is needed for the PERL scripts
anchor snort2pf
#
```

Also take a look at these two excellent pf sites:

- http://www.benzedrine.cx/pf.html
- https://www.solarflux.org/pf/ - Many of the ideas for this practical were inspired by the links found on solarflux, specifically Snort2pf and SnortSam.

Step 5

Step 5 does not involve too much security configuration. You will install Snort, Nessus, and the PERL scripts. Remember to add any important file to the /etc/changelist for backup. For binary files just add a + in front of the path and filename to indicate that you only want an md5sum of the file backed up, as in "+/usr/local/sbin/nessusd".

First, reboot the server. Use nmap from other machines to verify that your pf rules are actually working the way you intended. Looking at both the output of nmap and the output of "/usr/bin/netstat –na" verify that no daemons other than those that you intended are still running. Once you have ensured that the server is configured correctly, connect it to the network. The process of installing Snort and Nessus in chroot-ed environments will not be detailed or performed in this example, as it is not deemed necessary given the role of this server. It is however a recommended best practice and is possible with both the Snort and Nessus daemons.

The order you install Snort and Nessus is not important. Nessus has at least one compile time option that has security implications. For nessus-core use "./configure --disable-gtk" before make. As an alternative to compiling your own you might want to try the OpenBSD package nessus-2.0.9-no_x11.tgz. Note however that the current version is Nessus 2.0.10a. If you downloaded the sources from the web, as always, verify the md5sum of the tgz files with at least one server that is not the server from which you downloaded the files.

Copy the three PERL scripts from Appendix C to files in your /usr/local/sbin folder. The permissions on those files should be group wheel owned by root and 0760. Actually, if special members of your administrative team are usually responsible for maintaining scripts, create a special group for them using /usr/sbin/groupadd, add their normal user accounts to that group using /usr/sbin/usermod, and make these files belong to that group instead with /usr/sbin/chown. Verify the sanity of the permissions of the other files in /usr/local/sbin at this point also. Does world really need read permissions on nessusd? Given that you do not have normal users, think critically about all of the

default file and directory permissions on the partitions /var, /usr, /tmp, and /. Once you have everything configured the way you like it, add the settings to the /etc/mtree/special file and the /etc/security script that is run daily will let you know if any of these permissions get changed.

Step 6

In step 6 you will configure Snort to alert when it detects traffic that violates your site's acceptable network usage policy. Basic Snort configuration requires that you add and make changes to the snort.conf and Snort rules files as well as add it to your /etc/rc.local script to start it on boot. You will also need to add the snort directory to /var/log/ for its alert file.

To get the current rules, go to http://www.snort.org/dl/rules/. There are three rules files available and you need to make sure that you get the right one based on the version of the source you compiled. As with everything you download, check the md5. Also note the file date. The rules files are usually updated daily, adding rules for the latest threats. Once downloaded, uncompress the rules file into the folder /etc/snort/rules.

First open the file snort.conf in /etc/snort/rules. The file is very well commented and is used to set the values for variables that will be referenced in all of the .rules files. In addition to the variables you also need to configure the preprocessors as well. Remember that the escorting firewall is not an Internet connected system and as such is less likely to trigger false positives even with the default rules. The most important lines are "var HOME_NET 192.168.6.0/24" and "var EXTERNAL_NET any". Snort will only alert on attacks against addresses specified by HOME_NET from systems that match EXTERNAL_NET. Note that the default any value for EXTERNAL_NET will alert on attacks within your subnet, which is exactly what we need for this example. Since you will be configuring white lists in snort2pf it isn't as critical to remove trusted addresses from the EXTERNAL_NET but in the interest of saving processor cycles you might want to anyway. The cost of doing so is that should one of the trusted systems be compromised you will not even be able to refer to the Snort log to see what the trusted machine had been doing. In this example the following addresses on the 192.168.6.0/24 network are to be trusted no matter what .105, .22, .69, and .1. To configure EXTERNAL_NET to avoid processing and alerting on these, use the following:

"var EXTERNAL_NET [!192.168.6.105/32,!192.168.6.22/32,!192.168.6.69/32,!192.168.6.1/32]"

In addition to the default preprocessors consider enabling portscan. Carefully read the portscan comments however as it behaves a little differently. Snort2pf has regular expressions to match portscan alerts. Also, don't let the portscan.log at the end of the line fool you, portscan will still alert as normal in the /var/log/snort/alert file, the portscan.log file simply exists to show the actual portscan details. Finally, add the following to the end of your /etc/rc.local file, obviously you may need to adjust the –i value:

```
# Snort stuff
if [ -x /usr/local/bin/snort ]; then
        echo -n ' snort';       /usr/local/bin/snort -D -A FULL -i fxp0 -c /etc/snort/rules/snort.conf
fi
```

If you are very familiar with your network traffic and with configuring Snort rules, you may want to make some adjustments now. In this example, the default rules are fine. The assumption being made is that the default Snort rules will enforce the site policy. In most cases, the default Snort rules will enforce only parts of the acceptable network usage policy and there is a small chance that some authorized uses might mistakenly be identified as policy violations. One important point to be aware of is that this setup is using a tool designed to detect attacks, to detect acceptable network usage policy violations. If your policy defines the use of telnet as a policy violation, you wouldn't want to alert only on "TELNET login incorrect" or "TELNET Solaris memory mismanagement exploit attempt", you would want to alert on every attempted use of telnet. Fortunately, Snort is still the right tool for the job, but you will need to write your own Snort rules. While writing Snort rules is beyond the scope of this practical read any of the many great SANS GIAC GCIA pracitcals for a better introduction to Snort.

Step 7

In this step you will configure the Nessus daemon and PERL scripts. Nessus configuration is much like the Snort configuration; you will configure a conf file, a rules file and add the daemon to the /etc/rc.local file. The major difference is that to configure Nessus you will also need to add a Nessus user account for your script. The PERL scripts are trickier because a high amount of customization may be required to get them to work in your environment. Syslog will need to be configured to get the scripts to work properly in this instance. Reread the specification, it determines how the scripts are supposed to function.

Adding a user for Nessus is actually the first thing you need to do when configuring Nessus. Use the command /usr/local/sbin/nessus-adduser to create your Nessus user. Remember the password recommendations suggested earlier? Ignore them. You're going to write this password down in a script and never need to remember it. It is to your advantage to make this password the longest and most obnoxious password possible that will not require using escape characters to type in. Of course, using escape characters might be an easy way to fool novice attackers that manage to get access to the script source. I was able to create a 130-character long password. I then tried a 260-character length password and though there was not an error when I created the account, I always received login failed when I tried to use it. When you are adding Nessus user accounts you may want to configure rules for the account. If the only user account will be for the script, then wait to make the changes in the nessusd.rules file.

Before configuring the /usr/local/etc/nessus/nessusd.conf file, add your rules to your nessusd.rules file. The format for adding rules in the rules file is the same as adding rules for users. Read the man page for instructions or follow the rules file used in this example:

```
# Nessus rules
#
# Syntax : accept|reject address/netmask
```

```
# Deny servers we never want to allow scans against
reject 192.168.6.1/32
reject 192.168.6.69/32
reject 192.168.6.22/32
reject 192.168.6.105/32
# Allow all other systems on the network
accept 192.168.6.0/24
# Deny by default :
default reject
```

The conf file for Nessus is in the /usr/local/etc/nessus folder and is named nessusd.conf. If you do not have a file already, you will after you start Nessud the first time. The lines that have a significant bearing on security are detailed here:

```
# Path to the security checks folder :
plugins_folder = /usr/local/lib/nessus/plugins

# Maximum number of simultaneous hosts tested (the script will only attempt to scan 1 at a time):
#max_hosts=10
max_hosts = 1

# Niceness. If set to 'yes', nessusd will renice itself to 10. (not even this value is nice enough if an attacker is able
#       to submit more than fifty consecutive scans)
#be_nice=no
be_nice = yes

# Log file (or 'syslog') (you must change this to something on /var if you want to mount /usr read-only later):
#logfile = /usr/local/var/nessus/logs/nessusd.messages
logfile=syslog

# Shall we log every details of the attack (good chance you will want to change this to no to save disk space)?
log_whole_attack = yes

# Dump file for debugging output, use `-' for stdout (again, /usr might get mounted readonly)
# dumpfile = /usr/local/var/nessus/logs/nessusd.dump
dumpfile = /var/log/nessusd.dump

# Read timeout for the sockets of the tests (you are scanning remote access users that might potentially be on high
latency connections, give them time to respond):
# checks_read_timeout =5
checks_read_timeout = 15
```

Lastly you need to edit /etc/rc.local and add the following lines after the Snort lines you added in step 6, the –a 127.0.0.1 binds the daemon to the loopback address:

```
# Nessus stuff
if [ -x /usr/local/sbin/nessusd ]; then
        echo -n ' nessusd';     /usr/local/sbin/nessusd -a 127.0.0.1 -D
fi
```

Before you just decide to run the included PERL scripts, take a minute to read them and familiarize yourself with each line. These scripts were not intended to run on a production server but merely to serve as a proof of concept. The PERL script Snort2pf was written by Stephan Schmieder http://bsd-security.org/~ssc/codedocs/snort2pf/. With much help from Gerald Comeaux GSEC #3060, http://www.giac.org/GSEC_3100.php, I modified the Snort2pf script to include support for white lists and to interoperate with an entirely new

script, conceptually based on Snort2pf, which watches for new remote access connections and starts Nessus scans against them. Gerald did not actually want to be credited due to the very proof of concept nature of the scripts. Essentially, Snort2pf tails the Snort alert file. When Snort logs an attack, Snort2pf takes the address from the alert and dynamically configures pf to block that host using the anchor you setup in the pf.conf file.

Scanaccess, the aforementioned second script, operates in a similar manner. Rather than monitoring the Snort alert file however, scanaccess tails a syslog file that receives messages from the Cisco PIX 501 firewall and VPN remote access device. When scanaccess detects a new user it calls a third script NessusScan. NessusScan launches nessus scans against the VPN user's VPN-assigned IP address. Obviously, these scripts are highly environment specific. Unless, you have a 500 series Cisco Pix, with a very specific vpn configuration that includes RADIUS authentication to be enabled for IKE phase 1, with matching vpngroup group and RADIUS user names, you will have to make changes.

Before continuing please note, in many remote access situations a pool of addresses is assigned to a group of remote access users. If that is your setup, you will need to spend some time reengineering both the Snort2pf and the scanaccess scripts. These scripts are not really user aware. Snort2pf instructs pf to block IP addresses not users. In the event that a remote access user violating acceptable network usage policy is detected by Snort and then blocked by pf, it is the IP address of the remote access user not the user itself that is blocked. In the case where a pool of addresses is assigned to a group of users, when the attacking user disconnects and then reconnects it may be possible for them to get a different IP address. As the scripts are currently written, the policy-violating user will no longer be blocked because they have a new IP address. Potentially worse, is that when a user that has not violated connects, if they are assigned an address that is still being blocked, the legitimate user will be denied access.

To prevent this, the remote access device in the test network has been configured to assign a single unique IP address for each remote access user. If you do not have enough addresses to accommodate this model or if you do not want the additional administrative burden of keeping track of which user is assigned which IP address, you will need to write a script that maintains a state table mapping user accounts with their currently assigned IP address. Each time a user connects this new script will need to update the state table and if necessary, pf, and the blocked_hosts file also.

Make sure to create a whitelist file that contains the IP addresses of all of the systems that you do not ever want to block. You can put the file wherever you want as long as you update the script accordingly. The file should contain one IP address per line with no extra characters or white space. If you want to add comments to the file or make the file more tolerant of extraneous characters, then you will need to rewrite the init_white sub routine.

```
# /bin/cat /usr/local/sbin/snort2pf
#!/usr/bin/perl -T
# Addittions Copyright Benjamin Eason 2004 <beason@sec-res.com>
# Copyright (c) 2003, 2004
#     Stephan Schmieder <ssc@unix-geek.info>.  All rights reserved.
```

```
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
# 1. Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
# 2. Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in the
#    documentation and/or other materials provided with the distribution.
#
# THIS SOFTWARE IS PROVIDED BY STEPHAN SCHMIEDER AND CONTRIBUTORS ``AS IS'' AND
# ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
# ARE DISCLAIMED.  IN NO EVENT SHALL STEPHAN SCHMIEDER OR CONTRIBUTORS BE LIABLE
# FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
# DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
# OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
# OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
# SUCH DAMAGE.

use strict;

#use warnings;
#use diagnostics;

my ( $blockfile, $alertfile, $pfctl, $tail, $amnesty, %bad_hosts, @white_hosts, $tmp,
$alert_size, $block_size, $update, $line, $list, @array, $whiteFile, $cp );

# <configuration>
$alertfile = '/var/log/snort/alert'; # The Snort alert file
$blockfile = '/var/log/escort/blocked_hosts'; #This is just a temp file
$whiteFile = '/etc/escort/white_hosts'; # Put trusted systems in here 1 per line
$pfctl     = '/sbin/pfctl';
$tail      = '/usr/bin/tail';
$cp        = '/bin/cp';
$amnesty   = 60 * 60 * 12; # seconds * minutes * hours to wait before unblocking IP
# </configuration>
my $version = "Escort";

# Call the initialize sub routine
&initialize();

# After initialize, run this infinite loop that checks for Snort alerts and unblocks
#              based on the amnesty value
while (sleep(1)) {


   #unblock old hosts
   &unblock( time - $amnesty );

   # process alertfile if modified
   $tmp = (stat($alertfile))[7];
   if ($tmp > $alert_size) {
      open( DATA, "$tail -c ".($tmp - $alert_size)." $alertfile |" )
         or die("Unable to read last bytes from alertfile ($!)\n");
      $alert_size = $tmp;
      while ( defined( $tmp = <DATA> ) ) {
         chomp $tmp;
         if ($tmp) {
            &block( &check_for_attack( &check_for_portscan($tmp) ) );
         }
      }
      close(DATA);
   }
}

# The initialize sub reads in the white list and gets the intial size of the Snort Alert
#              file. It also processes the command line arguments.
sub initialize {

   # parse command line
   for ( my $i = 0; $i < $#ARGV; $i++ ) {
      if ( $ARGV[$i] eq '-s' ) {
         if ( $ARGV[++$i] =~ /^(?:\d+)$/ ) {
            $amnesty = $1;
         } else {
            die("insane option for \"-s\"");
         }
      } elsif ( $ARGV[$i] eq '-f') {
```

```
            if ( $ARGV[++$i] =~ /^([0-9A-z\.\/]+)$/ ) {
               $alertfile = $1;
            } else {
               die("insane option for \"-f\"");
            }
         }
      }

      # set/check environment
      delete @ENV{qw(PATH IFS CDPATH ENV)};
      &update_title();
      if ( !-e $blockfile ) {  #The blockfile was added to synchronize the two main scripts
         system("$cp /dev/null $blockfile"); #I don't like to touch
      }
      if ( !-r $alertfile ) {
         die("\"$alertfile\" is not readable");
      }
      if ( !-x $pfctl ) {
         die("\"$pfctl\" is not executeable");
      }
      if ( !-x $tail ) {
         die("\"$tail\" is not executeable");
      }
      $alert_size = ( stat($alertfile) )[7];
      $block_size = ( stat($blockfile) )[7];

      # Get whitelist, the list of systems to never block
      init_white();
   }

   sub init_white { #This sub loads a list of IP addresses from a file, 1 per line
      my ($white_addr,$i);
      $i=0;
      open (WHITE, "$whiteFile");
      while ($white_addr = <WHITE>) {
         chomp($white_addr);
         $white_hosts[$i] = $white_addr;
         $i+=1;
      }
      close (WHITE) || die("Unable to close white_hosts file ($!)");
   }

   sub update_title {
      # update the string that's shown by ps(1) - very cool feature
      my $hosts = scalar keys %bad_hosts;
      $0 = 'snort2pf '.$version.' :: blocking '.(scalar keys %bad_hosts).' hosts';
   }

   sub check_for_attack {
      if ( $_[0] =~
         /^(?:\d{2}(?:\/|-|:|\.)){5}\d{6} ((?:\d{1,3}\.){3}\d{1,3})[: ]/ )
      {
         $_[0] = $1;
      }
      return ( $_[0] );
   }

   sub check_for_portscan { #Portscan preprocessor not enabled by default, see snort.conf
      if ( $_[0] =~ /PORTSCAN DETECTED from ((?:\d{1,3}\.){3}\d{1,3})[: ]/i ) {
         $_[0] = $1;
      }
      return ( $_[0] );
   }

   sub block {

      # reset $update
      my $update = 0;

      # this hash is unique to this sub
      my %block_array;

      # validate IPv4 address
      my $tmp = $_[0];
      if ( $tmp =~ /^(\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})$/ ) {
         if (    $1 >= 1
            and $1 <= 255
            and $1 != 127
            and $2 >= 0
            and $2 <= 255
            and $3 >= 0
```

```
          and $3 <= 255
          and $4 >= 1
          and $4 <= 254
          and ! check_if_white($tmp) ) # Added check for whitelist
          {
             # block host
             $bad_hosts{$tmp} = time;
             # write this to a file
             open( BLOCK, "+< $blockfile");

             # Take the current file, read it in to a hash. $block_array[ip] time of block
             while ($line = <BLOCK>) {
                chomp($line);
                @array = split(/ /,$line);
                $block_array{$array[0]} = $array[1];
             }

             # If the ip is already in the list, don't add it again, just update the time
             foreach $list (keys (%block_array)) {
                if ($tmp eq $list) {
                   $block_array{$list} = time;
                   $update=1;
                }
             }

             # clear the file, and repopulate it with the current info from the hash
             seek(BLOCK,0,0);
             foreach $list (keys (%block_array)) {
                print BLOCK "$list $block_array{$list}\n";
             }
             truncate(BLOCK,tell(BLOCK));

             # finally, if we didn't update a time, add the ip and time to the end of the
             #          file
             if (!$update) {
                print BLOCK "$tmp $bad_hosts{$tmp}\n";
                print  "adding $tmp $bad_hosts{$tmp}\n";
             }
             close( BLOCK ) || die ("Can't close blockfile ($!)\n");

             # after it's been recorded in the file, do the actual blocking
             open( PFCTL, "| $pfctl -a snort2pf:$tmp -f -" )
                or warn("Can't block $tmp($!)\n");
             print PFCTL "block in quick from $tmp to any\n";
             close(PFCTL) or die("Can't write to pfctl pipe($!)\n");
          } # closing if=properly formatted and not white host
       } # closing if=proper numbers
       &update_title();
}  # closing sub

sub check_if_white {
   my ($passed_target) = @_;
   my ($other_white_addr);
   print "checking if whitelisted\n";
   foreach $other_white_addr (@white_hosts) {
      if ($other_white_addr eq $passed_target) {
         return 1;
      }
   }
   return 0;
}

sub unblock {
# idea: we don't want to be constantly reading from the drive every second.
# we need to have a way to keep the records in a hash (%bad_hosts?)
# and only update if the file's been changed, a la the original snort2pf

   # this hash is unique to this sub
   my %block_array;

   # process blockfile if modified

   $tmp = ( stat($blockfile) )[7];
   if ( $tmp > $block_size ) {
      open( DATA, "$tail -c ".($tmp - $block_size     )." $blockfile |")
         || die("Unable to read from blockfile ($!)\n");
      $block_size = $tmp;
      while ( defined( $tmp = <DATA> )) {
         chomp $tmp;
         # read it into an array again
         @array = split(/ /,$tmp);
```

```
            $block_array{$array[0]} = $array[1];
        }

        #after populating the %block_array, add new stuff to %bad_hosts
        #at this point, the file should always be the more up to date of the two
        foreach $list (keys %block_array) {
            if (!exists ($bad_hosts{$list})) {
                $bad_hosts{$list} = $block_array{$list};
            }
        }

        #finally, check for and perform unblocks,and update file
        seek(DATA,0,0);
        truncate(DATA,tell(DATA));
        foreach $tmp ( keys %bad_hosts ) {
            if ( $bad_hosts{$tmp} <= $_[0] ) {
                delete $bad_hosts{$tmp};
                !system("\"$pfctl\" -a snort2pf:$tmp -F rules")
                    or warn("Can't unblock $tmp($!)\n");
            } else {
                print DATA "$tmp $bad_hosts{$tmp}\n";
            }
        }

        close( DATA ) || die("Could not close blockfile ($!)\n");

    } #this closes the 'if' loop commented out above

    &update_title();
}
```

Before scanaccess will work, several configurations must be made. First, you will need to devise a way to get a notification when new users connect from your remote access server. In this example, the Cisco PIX was configured to send syslog messages to the escorting firewall. The file syslog.conf was configured to put local4 messages into a separate file, “local4.debug /var/log/ciscolog”. When a VPN user connects they are required to login via RADIUS to complete the initial key exchange, IPSec phase 1. Upon successful authentication the PIX sends a syslog message to the escorting firewall that includes the name of the successfully authenticated user. As mentioned earlier, each VPN user has an IP address assigned to them when they connect and that IP address never changes. Unfortunately, the PIX does not also include the IP address of the successfully authenticated user (actually the user has yet to receive its IP address which is the last step of phase 1). To account for this, the script relies on a file called userlist that enables the script to match user accounts with IP addresses. You will need to create this file. The scanaccess initialize function expects the userlist file to be in the following format:

```
    # /bin/cat /etc/escort/userlist
    Administrator
    192.168.6.200
    Barbara
    192.168.6.201
    Tina
    192.168.6.209
    John
    192.168.6.205
```

```
# /bin/cat /usr/local/sbin/scanaccess
#!/usr/bin/perl
# The -T parameter should be used for perl. This proof of concept script
#       sometimes however has difficulties with insecure system calls.
# Additions Copyright Benjamin Eason 2004 <beason@sec-res.com>
# Copyright (c) 2003, 2004
#     Stephan Schmieder <ssc@unix-geek.info>.  All rights reserved.
```

```
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
# 1. Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
# 2. Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in the
#    documentation and/or other materials provided with the distribution.
#
# THIS SOFTWARE IS PROVIDED BY STEPHAN SCHMIEDER AND CONTRIBUTORS ``AS IS'' AND
# ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
# ARE DISCLAIMED.  IN NO EVENT SHALL STEPHAN SCHMIEDER OR CONTRIBUTORS BE LIABLE
# FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
# DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
# OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
# OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
# SUCH DAMAGE.


use strict;

#use warnings;
#use diagnostics;

my ( $username, $pfctl, $loginfile, $whiteFile, $tail, %bad_hosts, %user_list, $tmp,
$size, $name, $ip_address, $userlist, @fields, @white_hosts, $echo, $NessusScan,
$nessustargetpath );

# <configuration>
$loginfile = '/var/log/ciscolog';
$tail      = '/usr/bin/tail';
$pfctl     = '/sbin/pfctl';
$echo      = '/bin/echo';
$NessusScan= '/usr/local/sbin/NessusScan';
$userlist  = '/etc/escort/userlist';
$whiteFile = '/etc/escort/white_hosts';
$nessustargetpath = '/var/log/escort';
# </configuration>

&initialize();

while (sleep(1)) {


   # check diff in bytes, then read it in
   $tmp = ( stat($loginfile))[7];
   open( DATA, "$tail -c ".($tmp - $size)." $loginfile |" )
      or die("Unable to read last bytes from loginfile ($!)\n");
   # save that position for later
   $size = $tmp;
   while ( defined( $tmp = <DATA> ) ) {
      chomp ($tmp);
      if ($tmp) {
         @fields = split(/ /,$tmp);
         if (($fields[5] eq "Authen") and ($fields[7] eq "Start:")) {
            print "$fields[9]";
            $username = substr($fields[9],1,30);
            chop ($username);
            chop ($username);
            $ip_address = $user_list{$username};
            print "Username: $username\n";
            if (! check_if_white($ip_address)) {
               #nessus wants to read it's target from a file, so put it there.
               system("$echo $ip_address > $nessustargetpath/$ip_address-target");
               system("$NessusScan $ip_address &");
            } # End If not on white list
         }
      }
   }
   close(DATA);
}

sub initialize {

   # set/check environment
   delete @ENV{qw(PATH IFS CDPATH ENV)};
   if ( !-r $loginfile ) {
```

```
      die("\"$loginfile\" is not readable");
   }
   if ( !-x $tail ) {
      die("\"$tail\" is not executeable");
   }
   $size = ( stat($loginfile) )[7];

   open (USERLIST, "$userlist");
   while ($name = <USERLIST>) {
      chomp ($name);
      $ip_address = <USERLIST>;
      chomp ($ip_address);
      $user_list{$name} = $ip_address;
   }
   close (USERLIST);
   init_white();

}

sub init_white {
   my ($white_addr,$i);
   $i=0;
   open (WHITE, "$whiteFile");
   while ($white_addr = <WHITE>) {
      chomp($white_addr);
      $white_hosts[$i] = $white_addr;
      $i+=1;
   }
   close (WHITE) || die("Unable to close white_hosts file ($!)");
}

sub check_if_white {
   my ($passed_target) = @_;
   my ($other_white_addr);
   print "\nChecking if $passed_target is on whitelist\n";
   foreach $other_white_addr (@white_hosts) {
      if ($other_white_addr eq $passed_target) {
         print "$passed_target is on whitelist.\n";
         return 1;
      }
   }
   print "$passed_target is NOT on whitelist.\n";
   return 0;
}
```

NessusScan is called by scanaccess. NessusScan accepts an IP address as an argument and uses it to create the Nessus target and result files. NessusScan only accepts 1 IP address. This design decision was made in an attempt to multi-thread scanaccess. Because Nessus scans can take more than five minutes against systems connected on a LAN, you definitely do not want to wait for a scan to complete against a remote user. You will certainly have to change the $scan variable to include your Nessus username and password information, the port as well if you changed it from the default tcp 1241.

```
# /bin/cat /usr/local/sbin/NessusScan
#!/usr/bin/perl

use strict;

my ($holes_threshold, $warnings_threshold, $targetFile, $resultsFile, $blocked_hosts,
@bfields, $bfields, $line, $scan, $rm, $nessus, $rfields, @rfields, $time, $warnings );

$holes_threshold = 3;
$warnings_threshold = 7;
$targetFile  = "/var/log/escort/$ARGV[0]"."-target";
$resultsFile = "/var/log/escort/$ARGV[0]"."-results";
$blocked_hosts = "/var/log/escort/blocked_hosts";
$rm            = "/bin/rm";
$nessus        = "/usr/local/bin/nessus";
$scan = "$nessus -q 127.0.0.1 1241 escortscript thescriptpassword";

check_if_blocked();
run_scan();
```

```
check_results();

sub check_if_blocked {
   print "checking if blocked\n";
   open (BLOCKED, "$blocked_hosts")
      || die("Unable to open blocked_hosts file ($!)\n");
   while ($line = <BLOCKED>) {
      chomp ($line);
      @bfields = split(/ /,$line);
      if ($bfields[0] eq "$ARGV[0]") {
         system("rm $targetFile");
         die("$bfields[0] is already blocked\n");
      }
   }
   close (BLOCKED) || die("Unable to close blocked_hosts file ($!)");
}

sub run_scan {

   print "scanning $ARGV[0]\n";
   system("$scan $targetFile $resultsFile -T text > /dev/null");
   system("$rm $targetFile");

}

sub check_results {
   open ( RESULTS, "$resultsFile")
      || die("Unable to open results file ($!)\n");
   while (($line = <RESULTS>) && ($rfields[1] ne "HOSTS")) {
      chomp ($line);
      @rfields = split(/ /,$line);
      if ($rfields[5] eq "holes") {
         $holes = $rfields[8];
      }
      if ($rfields[5] eq "warnings") {
         $warnings = $rfields[8];
      }
   }

   close ( RESULTS ) || die("Unable to close results file ($!)\n");
   if (($holes ge $holes_threshold) || ($warnings ge $warnings_threshold)) {
      print "Host exceeds threshold, blocking\n";
      open(BLOCK, ">>blocked_hosts");
      $time = time;
      print BLOCK "$ARGV[0] $time\n";
      close( BLOCK ) || die("Can't close blockfile ($!)\n");
   }
}
```

Once all three scripts have been correctly configured for your environment you will need to add the two main scripts in the /etc/rc.local file. It is very important that both scripts are added after Snort and Nessus. Notice that the ampersands are required to send the scripts into the background. Without the ampersand Snort2pf would start but scanaccess and any boot up procedures that are to occur later would not until Snort2pf crashes or is killed.

```
   # Snort2pf stuff
   if [ -x /usr/local/sbin/snort2pf ]; then
               echo -n ' snort2pf';       /usr/local/sbin/snort2pf &
   fi

   # Scanaccess stuff, reads syslog and nessus scans VPN users
   if [ -x /usr/local/sbin/scanaccess ]; then
               echo -n ' scanaccess';       /usr/local/sbin/scanaccess &
   fi
```

Step 8

The final step in this step by step is verifying the correctness of the design and auditing the system. To analyze the correctness of this system you will want to review the specification written at the beginning of the project. Step 8 is continued in the last section of this paper entitled, Test & Verify Procedures.

Once your configuration is verified, you may want to make a complete backup image of the system in its known good state and then enable the advanced partition mounting options that were detailed in step 2.

Maintenance Procedures

The two keys of the strategy outlined in the risk mitigation plan were to reduce the threat surface of the escorting firewall and to quickly eliminate all known vulnerabilities. As important as making intelligent hardening decisions was to reducing the threat surface during setup, intelligent maintenance procedures are critical to eliminating vulnerabilities and maintaining security.

- Sign up for the mailing lists so you know when you need to patch.
  - OpenBSD security announce mailing list: mailto:majordomo@OpenBSD.org with a message body of "subscribe security-announce"
  - Snort announcement mailing list: http://lists.sourceforge.net/lists/listinfo/snort-announce
  - Nessus announcement mailing list: http://mail.nessus.org/mailman/listinfo/nessus-announce
  - Other mailing lists such as Full-Disclosure (high volume) http://lists.netsys.com/mailman/listinfo/full-disclosure
- Read root's messages every day, "/usr/bin/mail –u root" (or setup an alias to your normal user account).
- Read /var/log/failedlogin, /var/log/authlog, /var/log/secure, /var/log/snort/alert, /var/log/messages.
- Read /var/log/escort/blocked_hosts

Just as important as installing patches, is reading root's mail everyday. The security script that is run by /etc/daily, a cron job that runs daily, sends reports to root's mailbox. This author is willing to bet that if you've followed the suggestions in this hardening guide, three times out of five an attacker that has compromised the system will cause something out of the ordinary to show up in the daily security report and not delete it. If you do nothing else, follow the first two maintenance procedures. There really is not much that needs to be done. OpenBSD is very easy to manage and this is one of the reasons it was chosen to host the escorting firewall. The /etc/security script performs the following actions:

- Check the master passwd(5) and group(5) files for syntax, empty passwords, partially closed accounts, suspicious UIDs, suspicious GIDs, and duplicate entries.
- Check root's home directory and login environment for insecure permissions, suspicious paths, and umask commands in the dotfiles.
- Check that root and uucp are in /etc/ftpusers.
- Check for suspicious commands in /etc/mail/aliases.
- Check for insecurities in various trust files such as /etc/hosts.equiv, /etc/shosts.equiv, and /etc/hosts.lpd.
- Check user .rhosts and .shosts files for open access.
- Check user home directory permissions.
- Check many user dotfile permissions.
- Check user mailbox permissions.

- Check NFS exports(5) file for global export entries.
- Check for changes in setuid/setgid files and devices.
- Check disk ownership and permissions.
- Check for changes in the device file list.
- Check for permission changes in special files and system binaries listed in /etc/mtree/special and /etc/mtree/*.secure. Note: This is not complete protection against Trojan horsed binaries, as the miscreant can modify the tree specification to match the replaced binary. For details on really protecting yourself against modified binaries, see mtree(8).
- Check for content changes in those files specified by /etc/changelist. See changelist(5) for further details.

You might have noticed that absent from that list is a check for new network daemons. No problem. Create the file /etc/daily.local. Add the line "/bin/netstat -na | /usr/bin/egrep ^..p.+LISTEN > /var/log/networkports" Then add the file "/var/log/networkports" to the file /etc/changelist, and you will now know when your system is listening on new ports.

The daily and weekly cron jobs do some very cool things as well. /etc/daily

- Runs the script /etc/daily.local, if it exists.
- Removes scratch and junk files from /tmp and /var/tmp.
- Removes stale files from the rwhod(8) database.
- Checks for core dumps.
- Removes system messages older than 21 days for the msgs(1) utility.
- Purges accounting records from /var/account, if they exist. See accton(8) and sa(8).
- Creates a backup root file system which is updated daily. This only happens if the following conditions are met:
  1. ROOTBACKUP must be set to 1. It should be added to root's crontab(5):
     ROOTBACKUP=1
  2. The mount directory /altroot must exist, and there must be an /etc/fstab entry specifying `xx' for the mount options, e.g. /dev/wd0j /altroot ffs xx 0 0
- Checks disk status. Reports on the amount of disk used/available via df(1).
- Reports on which file systems need to be dumped via dump(8).
- Reports on the status of the mail queue via mailq(8).
- Reports networking statistics via netstat(1).
- Gives an uptime for every machine which exists in /var/rwho, via the ruptime(1) utility.
- Runs the calendar(1) utility unless the environment variable CALENDAR is set to 0 in root's crontab(5) or the host is a yp(8) client.
- If CHECKFILESYSTEMS is set to 1 in root's crontab, runs fsck(8) with the no-write flag (-n).

- If the file /etc/Distfile exists, runs the rdist(1) utility.
- Runs the system security check script, /etc/security. See security(8) for further details.

/etc/weekly

- Runs the script /etc/weekly.local, if it exists.
- Rebuilds the locate(1) database, if there is an existing /var/db/locate.database file.
- Rebuilds the whatis(1) database(s) via makewhatis(8).

Other than reading the messages generated by the cron scripts there are very few administrative tasks that cannot be scripted. For instance, if you want to make remote backups, just tgz the /var/backups folder and scp the file onto a remote file server. If it is annoying to read root's mail, you can have root's messages forwarded to one or more mailboxes by changing the line, "# root:" in the file /etc/mail/aliases and then running the command /usr/bin/newaliases.

Test & Verify Procedures

1. Root login locally
2. Root login through SSH
3. Nessus rules
4. scannaccess
5. NessusScan
6. Snort
7. snort2pf

To verify that the root cannot login to the locally, log out. Enter root for the username and then enter root's password. If your changes were successful, then you will get an error message that says, "." Next, look in the file /var/log/failedlogin and verify that failure appears there as well.

Root login through SSH should be denied. To try it now type "ssh root@127.0.0.1". It should be impossible to distinguish a mistyped password from a correctly typed password; you will get the message "Permission denied, please try again." In the /var/log/authlog file, a similar message in syslog will resemble, "Failed password for root from 127.0.0.1". If you were able to login, verify that "PermitRootLogin no" is not commented and is in your /etc/ssh/sshd_config file.

To verify that Nessus is configured and running properly, type "netstat –n". If netstat shows that the system is listening on *.1241, or *.whichever port you configured the Nessus daemon to run on, then you have a problem and need to check the –a argument in the /etc/rc.local file. If however the system only reports listening on 127.0.0.1.1241, then everything is okay. Still check to make sure that Nessus does not allow you to scan targets you did not explicitly write rules for it to accept. Create a target file that contains the IP address of a system that should never get scanned. Next using the –q argument to specify batch mode scanning, 127.0.0.1, your port number, your nessus username and password, the target file you just created, and some results file, run /usr/local/bin/nessus. Nessus will return no error and output nothing in the specified results file. This message will be placed in the file you specified for logfile in /usr/local/etc/nessus/nessusd.conf, "Date Time hostname nessusd: user username : rejected attempt to scan 192.168.6.69", where 192.168.6.69 is the address you entered in the target file. If the scan was successful, review your /usr/local/etc/nessus/nessus.rules file and ensure it is configured properly.

The scanaccess script should launch a Nessus against the IP address of the user that has connected. To verify that this is functioning properly, connect and login to the remote access server from a user account that does not match a whitelisted IP address. On the escorting firewall, run the "/bin/ps ax" command. In the out put you should see a line containing the following: "/usr/local/bin/nessus -q -T text 127.0.0.1 1241 /var/log/escort/192.168.6.200-target

/var/log/escort/192.168.6.200-results", where the IP address is the remote access server assigned IP address of the system from which you connected.

If the NessusScan script functioned properly there should be a log file when the scan is finished in /var/log/escort. The output of the log file should look similar to this one (some whitespace and superfluous new lines removed):

```
# /bin/cat /var/log/escort/192.168.6.200-results
Nessus Scan Report
------------------
SUMMARY

 - Number of hosts which were alive during the test : 1
 - Number of security holes found : 0
 - Number of security warnings found : 5
 - Number of security notes found : 5

TESTED HOSTS

 192.168.6.200 (Security warnings found)

DETAILS

+ 192.168.6.200 :
 . List of open ports :
   o ssh (22/tcp) (Security warnings found)
   o netbios-ssn (139/tcp) (Security notes found)
   o epmap (135/tcp)
   o general/tcp (Security warnings found)
   o general/udp (Security notes found)
   o general/icmp (Security warnings found)
   o netbios-ns (137/udp) (Security warnings found)

 . Warning found on port ssh (22/tcp)
    The remote SSH daemon supports connections made
    using the version 1.33 and/or 1.5 of the SSH protocol.

    These protocols are not completely cryptographically
    safe so they should not be used.

    Solution :
     If you use OpenSSH, set the option 'Protocol' to '2'
     If you use SSH.com's set the option 'Ssh1Compatibility' to 'no'

    Risk factor : Low

 . Information found on port ssh (22/tcp)
    An ssh server is running on this port

 . Information found on port ssh (22/tcp)
    Remote SSH version : SSH-1.99-OpenSSH_3.8p1

 . Information found on port ssh (22/tcp)
    The remote SSH daemon supports the following versions of the
    SSH protocol :
      . 1.33
      . 1.5
      . 1.99
      . 2.0

 . Information found on port netbios-ssn (139/tcp)
    An SMB server is running on this port

 . Warning found on port general/tcp
    The remote host uses non-random IP IDs, that is, it is
    possible to predict the next value of the ip_id field of
    the ip packets sent by this host.

    An attacker may use this feature to determine traffic patterns
    within your network.   A few examples (not at all exhaustive) are:

    1. A remote attacker can determine if the remote host sent a packet
    in reply to another request.  Specifically, an attacker can use your
    server as an unwilling participant in a blind portscan of another
    network.

    2. A remote attacker can roughly determine server requests at certain
```

```
    times of the day.  For instance, if the server is sending much more
    traffic after business hours, the server may be a reverse proxy or
    other remote access device.  An attacker can use this information to
    concentrate his/her efforts on the more critical machines.

    3. A remote attacker can roughly estimate the number of requests that
    a web server processes over a period of time.

    Solution : Contact your vendor for a patch
    Risk factor : Low

. Warning found on port general/tcp
    Your machine answers to TCP packets that are coming from a multicast
    address. This is known as the 'spank' denial of service attack.

    An attacker may use this flaw to shut down this server and
    saturate your network, thus preventing you from working properly.

    Solution : contact your operating system vendor for a patch.
               Filter out multicast addresses (224.0.0.0/4)

    Risk factor : Medium

. Information found on port general/udp
    For your information, here is the traceroute to 192.168.6.200 :
    192.168.6.22
    192.168.6.200

. Warning found on port general/icmp
    The remote host answers to an ICMP timestamp request. This allows an
     Attacker to know the date which is set on your machine.

    This may help him to defeat all your time based authentication protocols.

    Solution : filter out the ICMP timestamp requests (13), and the outgoing
     ICMP timestamp replies (14).

    Risk factor : Low
    CVE : CAN-1999-0524

. Warning found on port netbios-ns (137/udp)
    The following 2 NetBIOS names have been gathered :
     CHANGEME        = This is the computer name registered for workstation
     services by a WINS client.
     TEMP            = Workgroup / Domain name
    The remote host has the following MAC address on its adapter :
       0x00 0x05 0x9a 0x3c 0x78 0x00

    If you do not want to allow everyone to find the NetBios name
    of your computer, you should filter incoming traffic to this port.

    Risk factor : Medium
    CVE : CAN-1999-0621

------------------------------------------------------
This file was generated by the Nessus Security Scanner
#
```

If the number of holes or the number of warnings exceeded the threshold values that you specified in the /usr/local/sbin/NessusScan script, defaults 3 and 7 respectively, then you should also see the IP address in the /var/log/escort/blocked_hosts file. If you see the address in the /var/log/escort/blocked_hosts file but the thresholds were not exceeded, verify that the address was not blocked by Snort2pf, using the command "/bin/cat /var/log/snort/alert | /usr/bin/egrep 'PORT.+from 192.168.6.200|^.+:..:.+192.168.6.200:.+\-'". If it shows up, and the timestamps on the alerts are within the amnesty time, then it was most likely Snort2pf that blocked you. Also note that /var/log/messages should contain many lines pertaining to the Nessus scan if you configured /usr/local/etc/nessus/nessusd.conf to send its messages to syslog, otherwise they will be wherever you specified with the logfile value.

Assuming that you did not discover that your IP address had been blocked in the previous test, now you will want to test Snort. From a system that should not be on the whitelist, run an nmap stealth scan, -sS, against the escorting firewall with –T5 and –p 900-950 parameters. This will very very quickly scan TCP ports 900 through 950 on the escorting firewall. If you enabled the Snort portscan preprocessor, then running "/bin/cat /var/log/snort/alert | /usr/bin/egrep 'PORT.+from 192.168.6.200", where the IP address is the IP address from which you were running nmap, will produce a message such as the following: "[**] [100:1:1] spp_portscan: PORTSCAN DETECTED from 192.168.6.200 (THRESHOLD 5 connections exceeded in 0 seconds) [**]".

Finally, if Snort2pf was properly configured the IP address from which you were running nmap will appear in /var/log/escort/blocked_hosts. If the IP address does not appear there run "/bin/ps ax | /usr/bin/grep snort2pf". Check the number of hosts that Snort2pf believes that it is blocking. Verify that number against the number of IP addresses in /var/log/escort/blocked_hosts less the number of Nessus results files in /var/log/escort/ that exceed your NessusScan specified thresholds. Unfortunately with the current proof of concept scripts, there is no way to be certain whether or not Snort2pf and NessusScan both added the systems to the blocked_hosts file but you can compare the time for each host in the blocked_hosts file with the timestamps on the logs; this is admittedly a terrible procedure.

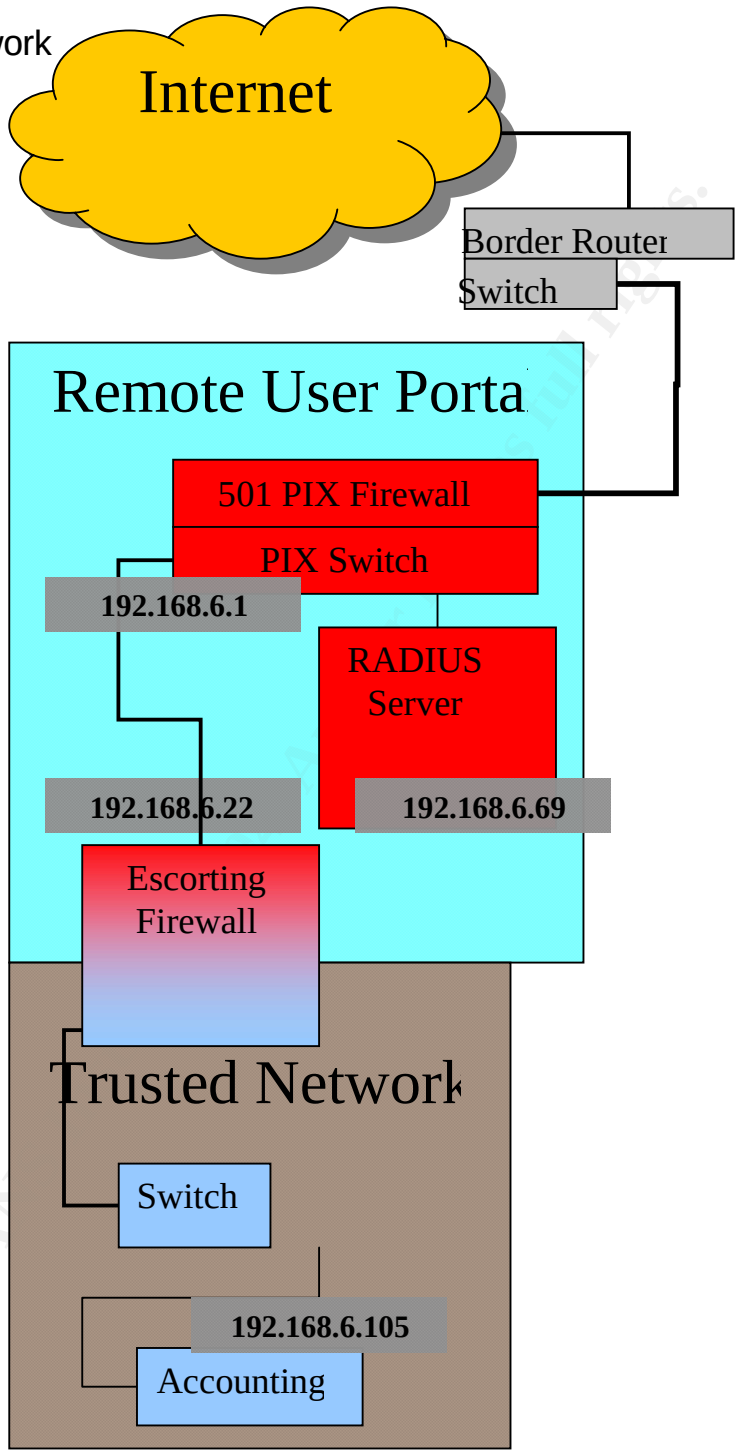
Conclusion

If you have read this far hopefully you have been inspired to rethink your approach to securely allowing remote users access to your network. This project was a lot fun writing about and hopefully the SANS community will benefit from this work. OpenBSD is a great platform for providing security services because it has been very well designed and because it is easy to maintain. PERL is easy to learn and is a good tool for automating security. If you have resisted automating your administrative tasks because you do not have a programming background and are intimidated by scripting, this is a great project to use to begin learning the PERL scripting language. Maybe the next feature you can add is a way to determine if your remote access users have the latest anti-virus definitions installed. Perhaps your policy forbids VPN split-tunneling, you could write a script to see if the user's public IP address responds to ICMP echo requests, or TCP connect scans. Whatever new problem you solve, be creative and add something useful to the community.

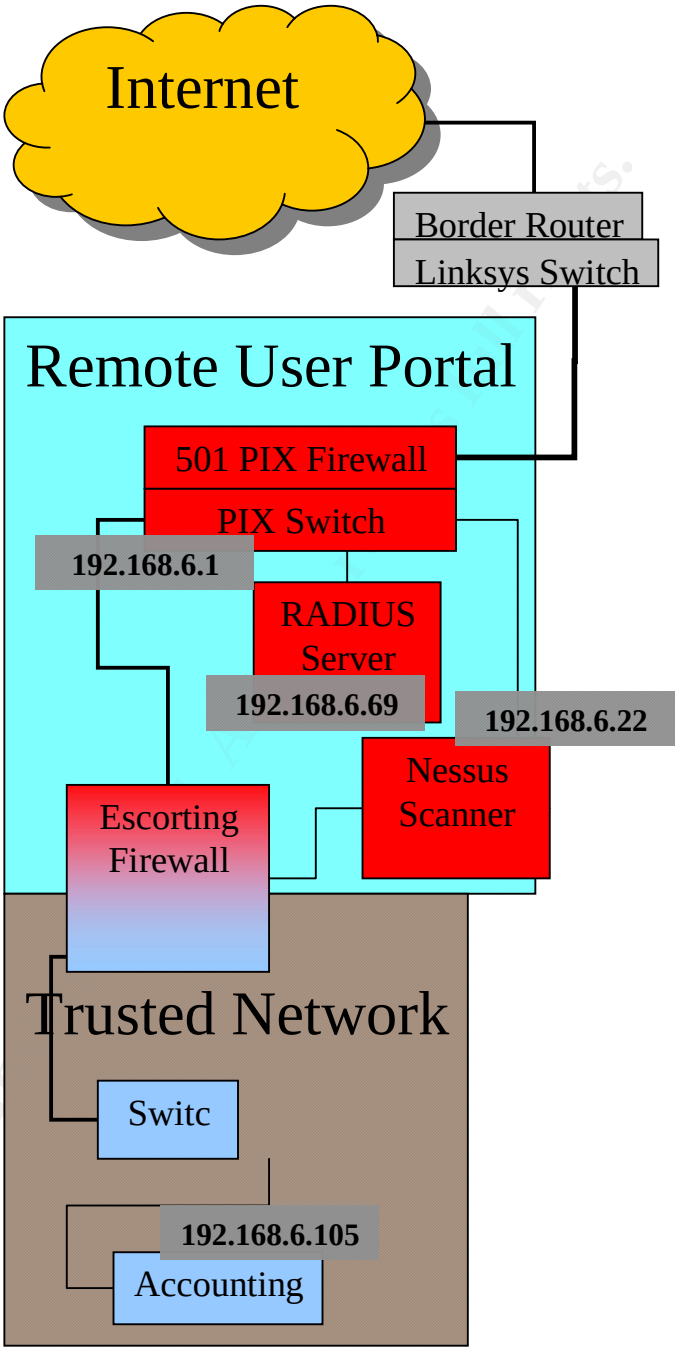
Appendices

Appendix A: Network Diagram

Example Network

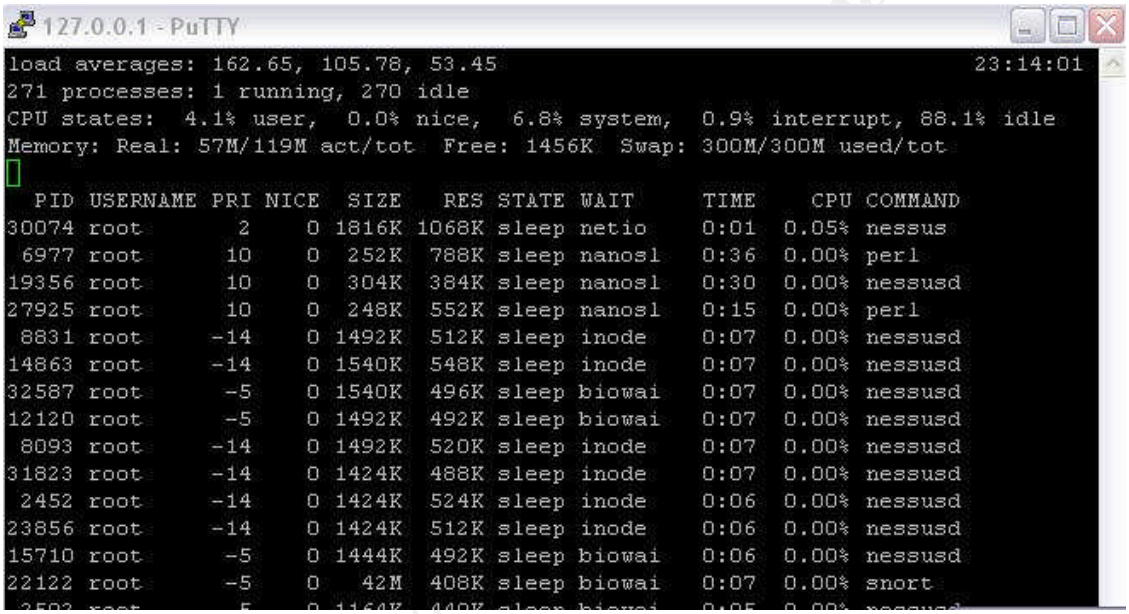


Transparent Design Model



Appendix B: System Under Fire

This image was taken during the vulnerability testing of Nessus. I started over 100 Nessus scans in an attempt to analyze how the system behaved during a resource attack. In this example I stopped sending new scan requests once the consoles became totally unresponsive. About ten hours later I checked the system and other than a few Nessus processes that had hung forever the system was fine. Later on I enabled the be_nice value in the nessus.conf file and generated twice as many scans. Ten hours later I power cycled the system because it still had not recovered. Hopefully you will only ever see this in a lab.



Appendix C: Proof of Concept PERL Scripts

Snort2pf – Written by Stephan Schmieder. Modified by Gerald Comeaux and Benjamin Eason

```
#!/usr/bin/perl -T
# Additions Copyright Benjamin Eason 2004 <beason@sec-res.com>
# Copyright (c) 2003, 2004
#     Stephan Schmieder <ssc@unix-geek.info>.  All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
# 1. Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
# 2. Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in
the
#    documentation and/or other materials provided with the
distribution.
#
# THIS SOFTWARE IS PROVIDED BY STEPHAN SCHMIEDER AND CONTRIBUTORS ``AS
IS'' AND
# ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
# ARE DISCLAIMED.  IN NO EVENT SHALL STEPHAN SCHMIEDER OR CONTRIBUTORS
BE LIABLE
# FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
# DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS
# OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY
# OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF
# SUCH DAMAGE.

use strict;

#use warnings;
#use diagnostics;

my ( $blockfile, $alertfile, $pfctl, $tail, $amnesty, %bad_hosts,
@white_hosts, $tmp, $alert_size, $block_size, $update, $line, $list,
@array, $whiteFile, $cp );

# <configuration>
$alertfile = '/var/log/snort/alert'; # The Snort alert file
$blockfile = '/var/log/escort/blocked_hosts'; #This is just a temp file
$whiteFile = '/etc/escort/white_hosts'; # Put trusted systems in here 1
per line
$pfctl     = '/sbin/pfctl';
$tail      = '/usr/bin/tail';
$cp        = '/bin/cp';
$amnesty   = 60 * 60 * 12; # seconds * minutes * hours to wait before
unblocking IP
# </configuration>
my $version = "Escort";
```

```
# Call the initialize sub routine
&initialize();

# After initialize, run this infinite loop that checks for Snort alerts
and unblocks
#           based on the amnesty value
while (sleep(1)) {


   #unblock old hosts
    &unblock( time - $amnesty );

   # process alertfile if modified
   $tmp = (stat($alertfile))[7];
   if ($tmp > $alert_size) {
      open( DATA, "$tail -c ".($tmp - $alert_size)." $alertfile |" )
        or die("Unable to read last bytes from alertfile ($!)\n");
      $alert_size = $tmp;
      while ( defined( $tmp = <DATA> ) ) {
        chomp $tmp;
        if ($tmp) {
          &block( &check_for_attack( &check_for_portscan($tmp) ) );
        }
      }
      close(DATA);
   }
}

# The initialize sub reads in the white list and gets the intial size
of the Snort Alert
#           file. It also processes the command line arguments.
sub initialize {

   # parse command line
   for ( my $i = 0; $i < $#ARGV; $i++ ) {
      if ( $ARGV[$i] eq '-s' ) {
        if ( $ARGV[++$i] =~ /^(?:\d+)$/ ) {
          $amnesty = $1;
        } else {
          die("insane option for \"-s\"");
        }
      } elsif ( $ARGV[$i] eq '-f') {
        if ( $ARGV[++$i] =~ /^([0-9A-z\.\/]+)$/ ) {
          $alertfile = $1;
        } else {
          die("insane option for \"-f\"");
        }
      }
   }

   # set/check environment
   delete @ENV{qw(PATH IFS CDPATH ENV)};
   &update_title();
   if ( !-e $blockfile ) {  #The blockfile was added to synchronize the
two main scripts
      system("$cp /dev/null $blockfile"); #I don't like to touch
   }
   if ( !-r $alertfile ) {
      die("\"$alertfile\" is not readable");
   }
   if ( !-x $pfctl ) {
      die("\"$pfctl\" is not executeable");
   }
```

```
   if ( !-x $tail ) {
     die("\"$tail\" is not executeable");
   }
   $alert_size = ( stat($alertfile) )[7];
   $block_size = ( stat($blockfile) )[7];

   # Get whitelist, the list of systems to never block
   init_white();
}

sub init_white { #This sub loads a list of IP addresses from a file, 1
per line
   my ($white_addr,$i);
   $i=0;
   open (WHITE, "$whiteFile");
   while ($white_addr = <WHITE>) {
     chomp($white_addr);
     $white_hosts[$i] = $white_addr;
     $i+=1;
   }
   close (WHITE) || die("Unable to close white_hosts file ($!)");
}

sub update_title {
   # update the string that's shown by ps(1) - Very cool feature
   my $hosts = scalar keys %bad_hosts;
   $0 = 'snort2pf '.$version.' :: blocking '.(scalar keys %bad_hosts).'
hosts';
}

sub check_for_attack {
   if ( $_[0] =~
     /^(?:\d{2}(?:\/|-|:|\.)){5}\d{6} ((?:\d{1,3}\.){3}\d{1,3})[: ]/ )
   {
     $_[0] = $1;
   }
   return ( $_[0] );
}

sub check_for_portscan { #Portscan preprocessor not enabled by default,
see snort.conf
   if ( $_[0] =~ /PORTSCAN DETECTED from ((?:\d{1,3}\.){3}\d{1,3})[: ]/i
) {
     $_[0] = $1;
   }
   return ( $_[0] );
}

sub block {

   # reset $update
   my $update = 0;

   # this hash is unique to this sub
   my %block_array;

   # validate IPv4 address
   my $tmp = $_[0];
   if ( $tmp =~ /^(\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})$/ ) {
     if (    $1 >= 1
       and $1 <= 255
       and $1 != 127
       and $2 >= 0
       and $2 <= 255
```

```
        and $3 >= 0
        and $3 <= 255
        and $4 >= 1
        and $4 <= 254
        and ! check_if_white($tmp) ) # Added check for whitelist
        {
           # block host
           $bad_hosts{$tmp} = time;
           # write this to a file
           open( BLOCK, "+< $blockfile");

           # Take the current file, read it in to a hash.
        $block_array[ip] time of block
           while ($line = <BLOCK>) {
              chomp($line);
              @array = split(/ /,$line);
              $block_array{$array[0]} = $array[1];
           }

           # If the ip is already in the list, don't add it again, just
        update the time
           foreach $list (keys (%block_array)) {
              if ($tmp eq $list) {
                 $block_array{$list} = time;
                 $update=1;
              }
           }

           # clear the file, and repopulate it with the current info from
        the hash
           seek(BLOCK,0,0);
           foreach $list (keys (%block_array)) {
              print BLOCK "$list $block_array{$list}\n";
           }
           truncate(BLOCK,tell(BLOCK));

           # finally, if we didn't update a time, add the ip and time to
        the end of the
           #       file
           if (!$update) {
              print BLOCK "$tmp $bad_hosts{$tmp}\n";
              print  "adding $tmp $bad_hosts{$tmp}\n";
           }
           close( BLOCK ) || die ("Can't close blockfile ($!)\n");

           # after it's been recorded in the file, do the actual blocking
           open( PFCTL, "| $pfctl -a snort2pf:$tmp -f -" )
              or warn("Can't block $tmp($!)\n");
           print PFCTL "block in quick from $tmp to any\n";
           close(PFCTL) or die("Can't write to pfctl pipe($!)\n");
        } # closing if=properly formatted and not white host
     } # closing if=proper numbers
     &update_title();
}  # closing sub

sub check_if_white {
   my ($passed_target) = @_;
   my ($other_white_addr);
   print "checking if whitelisted\n";
   foreach $other_white_addr (@white_hosts) {
      if ($other_white_addr eq $passed_target) {
         return 1;
      }
   }
```

```
   return 0;
}

sub unblock {
# idea: we don't want to be constantly reading from the drive every
second.
# we need to have a way to keep the records in a hash (%bad_hosts?)
# and only update if the file's been changed, a la the original
snort2pf

   # this hash is unique to this sub
   my %block_array;

   # process blockfile if modified

   $tmp = ( stat($blockfile) )[7];
   if ( $tmp > $block_size ) {
      open( DATA, "$tail -c ".($tmp - $block_size )." $blockfile |")
         || die("Unable to read from blockfile ($!)\n");
      $block_size = $tmp;
      while ( defined( $tmp = <DATA> )) {
         chomp $tmp;
         # read it into an array again
         @array = split( /,$tmp);
         $block_array{$array[0]} = $array[1];
      }

      #after populating the %block_array, add new stuff to %bad_hosts
      #at this point, the file should always be the more up to date of
   the two
      foreach $list (keys %block_array) {
         if (!exists ($bad_hosts{$list})) {
            $bad_hosts{$list} = $block_array{$list};
         }
      }

      #finally, check for and perform unblocks,and update file
      seek(DATA,0,0);
      truncate(DATA,tell(DATA));
      foreach $tmp ( keys %bad_hosts ) {
         if ( $bad_hosts{$tmp} <= $_[0] ) {
            delete $bad_hosts{$tmp};
            !system("\"$pfctl\" -a snort2pf:$tmp -F rules")
               or warn("Can't unblock $tmp($!)\n");
         } else {
            print DATA "$tmp $bad_hosts{$tmp}\n";
         }
      }

      close( DATA ) || die("Could not close blockfile ($!)\n");

   } #this closes the 'if' loop commented out above

   &update_title();
}
```

scanacces – Written by Gerald Comeaux and Benjamin Eason. Inspired by Stephan Schmieder's Snort2pf.

```
#!/usr/bin/perl
# The -T parameter should be used for perl. This proof of concept
script
#     sometimes however has difficulties with insecure system calls.
# Additions Copyright Benjamin Eason 2004 <beason@sec-res.com>
# Copyright (c) 2003, 2004
#     Stephan Schmieder <ssc@unix-geek.info>.  All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
# 1. Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
# 2. Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in
the
#    documentation and/or other materials provided with the
distribution.
#
# THIS SOFTWARE IS PROVIDED BY STEPHAN SCHMIEDER AND CONTRIBUTORS ``AS
IS'' AND
# ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
# ARE DISCLAIMED.  IN NO EVENT SHALL STEPHAN SCHMIEDER OR CONTRIBUTORS
BE LIABLE
# FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
# DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS
# OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY
# OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF
# SUCH DAMAGE.


use strict;

#use warnings;
#use diagnostics;

my ( $username, $pfctl, $loginfile, $whiteFile, $tail, %bad_hosts,
%user_list, $tmp, $size, $name, $ip_address, $userlist, @fields,
@white_hosts, $echo, $NessusScan, $nessustargetpath );

# <configuration>
$loginfile = '/var/log/ciscolog';
$tail      = '/usr/bin/tail';
$pfctl     = '/sbin/pfctl';
$echo      = '/bin/echo';
$NessusScan= '/usr/local/sbin/NessusScan';
$userlist  = '/etc/escort/userlist';
$whiteFile = '/etc/escort/white_hosts';
$nessustargetpath = '/var/log/escort';
# </configuration>
```

```
&initialize();

while (sleep(1)) {


   # check diff in bytes, then read it in
   $tmp = ( stat($loginfile))[7];
   open( DATA, "$tail -c ".($tmp - $size)." $loginfile |" )
     or die("Unable to read last bytes from loginfile ($!)\n");
   # save that position for later
   $size = $tmp;
   while ( defined( $tmp = <DATA> ) ) {
     chomp ($tmp);
     if ($tmp) {
       @fields = split(/ /,$tmp);
       if (($fields[5] eq "Authen") and ($fields[7] eq "Start:")) {
         print "$fields[9]";
         $username = substr($fields[9],1,30);
         chop ($username);
         chop ($username);
         $ip_address = $user_list{$username};
         print "Username: $username\n";
         if (! check_if_white($ip_address)) {
           #nessus wants to read it's target from a file, so put it
         there.
           system("$echo $ip_address > $nessustargetpath/$ip_address-
         target");
           system("$NessusScan $ip_address &");
         } # End If not on white list
       }
     }
   }
   close(DATA);
}

sub initialize {

   # set/check environment
   delete @ENV{qw(PATH IFS CDPATH ENV)};
   if ( !-r $loginfile ) {
     die("\"$loginfile\" is not readable");
   }
   if ( !-x $tail ) {
     die("\"$tail\" is not executeable");
   }
   $size = ( stat($loginfile) )[7];

   open (USERLIST, "$userlist");
   while ($name = <USERLIST>) {
     chomp ($name);
     $ip_address = <USERLIST>;
     chomp ($ip_address);
     $user_list{$name} = $ip_address;
   }
   close (USERLIST);
   init_white();

}

sub init_white {
   my ($white_addr,$i);
   $i=0;
   open (WHITE, "$whiteFile");
   while ($white_addr = <WHITE>) {
```

```
      chomp($white_addr);
      $white_hosts[$i] = $white_addr;
      $i+=1;
   }
   close (WHITE) || die("Unable to close white_hosts file ($!)");
}

sub check_if_white {
   my ($passed_target) = @_;
   my ($other_white_addr);
   print "\nChecking if $passed_target is on whitelist\n";
   foreach $other_white_addr (@white_hosts) {
      if ($other_white_addr eq $passed_target) {
         print "$passed_target is on whitelist.\n";
         return 1;
      }
   }
   print "$passed_target is NOT on whitelist.\n";
   return 0;
}
```

NessusScan – Written by Gerald Comeaux and Benjamin Eason. Inspired by Stephan Schmieder's Snort2pf

```
#!/usr/bin/perl

use strict;

my ($holes_threshold, $warnings_threshold, $targetFile, $resultsFile,
$blocked_hosts, @bfields, $bfields, $line, $scan, $rm, $nessus,
$rfields, @rfields, $time, $warnings );

$holes_threshold = 3;
$warnings_threshold = 7;
$targetFile  = "/var/log/escort/$ARGV[0]"."-target";
$resultsFile = "/var/log/escort/$ARGV[0]"."-results";
$blocked_hosts = "/var/log/escort/blocked_hosts";
$rm            = "/bin/rm";
$nessus        = "/usr/local/bin/nessus";
$scan = "$nessus -q 127.0.0.1 1241 escortscript thescriptpassword";

check_if_blocked();
run_scan();
check_results();

sub check_if_blocked {
   print "checking if blocked\n";
   open (BLOCKED, "$blocked_hosts")
      || die("Unable to open blocked_hosts file ($!)\n");
   while ($line = <BLOCKED>) {
      chomp ($line);
      @bfields = split(/ /,$line);
      if ($bfields[0] eq "$ARGV[0]") {
         system("rm $targetFile");
         die("$bfields[0] is already blocked\n");
      }
   }
   close (BLOCKED) || die("Unable to close blocked_hosts file ($!)");
}

sub run_scan {

   print "scanning $ARGV[0]\n";
   system("$scan $targetFile $resultsFile -T text > /dev/null");
   system("$rm $targetFile");

}

sub check_results {
   open ( RESULTS, "$resultsFile")
      || die("Unable to open results file ($!)\n");
   while (($line = <RESULTS>) && ($rfields[1] ne "HOSTS")) {
      chomp ($line);
      @rfields = split(/ /,$line);
      if ($rfields[5] eq "holes") {
         $holes = $rfields[8];
      }
      if ($rfields[5] eq "warnings") {
         $warnings = $rfields[8];
      }
   }

   close ( RESULTS ) || die("Unable to close results file ($!)\n");
   if (($holes ge $holes_threshold) || ($warnings ge
$warnings_threshold)) {
      print "Host exceeds threshold, blocking\n";
```

```
        open(BLOCK, ">>blocked_hosts");
        $time = time;
        print BLOCK "$ARGV[0] $time\n";
        close( BLOCK ) || die("Can't close blockfile ($!)\n");
    }
}
```

References

Felton, Ed. “State-Level "Super DMCA" Initiatives Archive: Affected States.” Electronic Frontier Foundation. 2003. URL: http://www.eff.org/IP/DMCA/states/#affectedstates (2 May 2004).

Holland, Nick. “Documentation and Frequently Asked Questions.” OpenBSD. 30 April 2004. URL: http://www.openbsd.org/faq/index.html (2 May 2004).

Schmieder, Stehpan. “Snort2pf.” SSC. 10 April 2004. URL: http://bsd-security.org/~ssc/codedocs/snort2pf/ (2 May 2004).

Shaffer, George. "Hardening OpenBSD Internet Servers." Geodsoft. 15 Dec 2001. URL: http://geodsoft.com/howto/harden/ (2 May 2004).

Slow2Show. “Mini-HOWTO on setting up Win2k, OpenBSD, and Linux for a triple boot system.” Gainesville2600. September 2002. URL: http://www.gainesville2600.org/stuff/obsd_linux_win2k_howto.htm

Last Updated: July 25th, 2017

Upcoming Training

SANS Network Security 2017	**Las Vegas, NV**	**Sep 10, 2017 - Sep 17, 2017**	**Live Event**
SANS OnDemand	**Online**	**Anytime**	**Self Paced**
SANS SelfStudy	**Books & MP3s Only**	**Anytime**	**Self Paced**