High Availability Firewalls using OpenBSD pf, pfsync and CARP

Athabasca University
MSc-IS Program
COMP503: Information Technology Hardware and Software

Allen Pomeroy
ID: 8702235

March 11, 2006

1. Management Overview

With the importance of the firewall as a key security tool to enforce information security policy, most organizations rely on commercial offerings which provide stateful connection control, network and port address translation (NAT/PAT), and in many cases built in intrusion detection (IDS) or intrusion prevention (IPS) features. These critical security devices present a single point of failure in an organizations connection to the Internet, extranets, and partitioned internal networks, yet since redundancy options in most commercial products are expensive, many organizations choose not to implement any form of redundancy at this network layer. The use of lower service options, such as vendor service level agreements, and even cold sparing where standby parts are kept on the customer site, still result in loss of network connectivity when the security device fails. With the increased dependence on network connectivity for geographically diverse communication and collaboration, failure of a security control device could have a significant negative financial impact due to lost productivity or missed opportunities.

The ideal outcome during a firewall failure is to have an alternate firewall take over the extended configuration of the failed firewall, and allow all existing connections to continue as if nothing had failed. To accomplish this, all existing connections must be maintained and new connections must be subject to the same security and routing policy that existed on the primary firewall.

This paper will discuss the business level design requirements for a highly available firewall service, an overview of the technical components used in the overall architecture of a solution, and a detailed analysis of the components used in the solution. The use of open source technology will be explored, as an effective replacement for commercial systems costing many tens of thousands of dollars, enabling the use of high availability technology on an economically inexpensive basis. To provide a complete solution using open source technology, multiple projects will need to be integrated.

Discussion of the OpenBSD operating system, pf packet filter, pfsync/CARP high availability components, and FWBuilder GUI project will illustrate a viable alternative to commercial firewall implementations costing several tens of thousands of dollars, bringing highly available firewall services to home based and SMB organizations.

Table of Contents

Contents

1. Management Overview 2
2. Introduction 3
2.1. Technical Problem 3
2.2. Business Requirements 3
2.3. Target Service Levels 3
3. Designing the Solution 3
3.1. Overall Design Assumptions 3
3.2. Security Policy Enforcement 3
3.3. High Availability 3
3.3.1. IP and MAC Address Failover 3
3.3.2. Configuration of CARP 3
3.3.3. pfsync for Connection State Table Synchronization 3
3.4. Management 3
3.5. Implementation 3
3.5.1. Hardware Used 3
3.5.2. Server Configuration 3
3.5.3. Packet Filter Logging 3
4. Solution Benefit Summary 3
A. References 3
B. Operating System Security 3
C. Network Interface Redundancy 3

Tables

Table 1 - Business Requirements - Protocols 3
Table 2 - fw1 Interface Assignments 3
Table 3 - fw2 Interface Assignments 3
Table 4 - CARP Interface Definitions (`/etc/hostname.carpx`) 3

Figures

Figure 1 - CARP and pfsync Filter Rules 3
Figure 2 - Firewall Architecture 3
Figure 3 - FWBuilder GUI Interface 3
Figure 4 - Interface Policy 3
Figure 5 - Firewall NAT/PAT Rules 3
Figure 6 - Firewall Configuration in FWBuilder 3
Figure 7 - Firewall Cables and Cable Harness 3
Figure 8 – Logical Network Diagram 3
Figure 9 - Security Manager Log Data 3

Listings

Listing 1 - fw1 CARP configuration 3
Listing 2 - fw2 CARP configuration 3
Listing 3 - Primary firewall interface state (fw1) 3
Listing 4 - Backup firewall interface state (fw2) 3

2. Introduction

2.1. Technical Problem

Many organizations have increased their reliance on firewall technology to block unwanted traffic, even between internal organization networks. Network firewalls are used to ensure only authorized traffic is allowed to enter or leave the networks that the firewalls protect. Firewalls can operate as either routers or bridges filtering packets against a set of rules that comprise the security policy that an owner has chosen to enforce. By applying a security policy to the packets attempting to traverse a network, the firewall reduces the exposure of the protected network segment to potentially malicious or unauthorized traffic.

There are two main techniques used by firewalls to enforce policy. Simple packet filtering is used to deny access to certain services, but its use is usually limited to protecting external network components such as routers or switches[1] since it cannot be used to track the complex relationships seen in upper layer protocols. Packet filtering operates on individual packets without regard for previous or subsequent packets that may be seen. Only information from the packet being evaluated is used to make a decision on the packet disposition, and typically only includes examination of the packet header information. Common uses include deployment on routers or other network perimeter devices to give protection from IP address spoofing[2] or denial of service (DoS) attacks[3]. The use of packet filtering does not introduce a single point of failure itself, as the packet filtering code does not keep any state on packets that it has allowed or dropped, and can be deployed on multiple devices to ensure alternate network paths are available to provide service should one network component fail.

The second technique used by firewalls to enforce security policy is stateful or dynamic packet filtering. Stateful packet filtering evaluates a packet based on header information, connection contextual information, and possibly deep inspection of the packet payload, to make a determination of whether to pass or drop the packet. Once evaluation of a rule set has resulted in the decision to pass a packet, information about that packet is inserted into an in memory connection state table. With the addition of the connection state table, filtering decisions are based on both firewall rule sets, and context built by packets that have passed through the firewall previously. To maintain connection context information, stateful filtering techniques manage this dynamic table of information about connections and related flows of traffic, inserting and removing entries from this connection state table as authorized connections are built up and torn down. As packets arrive at the policy enforcement point, header information in the packet is checked against the state table. If the packet matches an established connection in the state table, further policy evaluation is bypassed, the packet is forwarded, and the connection table is updated. Use of the state table to maintain connection information allows the firewall to also track connectionless protocols such as UDP and ICMP. If the packet does not match any entry in the state table, it is checked against the rule set contained on the firewall. The disposition of the packet is controlled by the results of checking the rule set, including inserting an entry in the connection state table if the packet matches a rule allowing passage. This use of an in memory connection state table to screen packets is significantly faster than checking packets against a rule set[4].

Although the use of a connection state table significantly increases the protection afforded by the firewall, it also introduces complications, which need to be addressed, in order to mitigate the single point of failure

---

[1] Router Security Configuration Guide, National Security Agency, December 2005, http://www.nsa.gov/snac/routers/C4-040R-02.pdf, pg 39

[2] Packet spoofing is defined as packets with source addresses that are improbable, such as a packet with a source address of an internal network arriving inbound at an external network interface. [add sans/nist definition]

[3] Router Security Configuration Guide, National Security Agency, December 2005, http://www.nsa.gov/snac/routers/C4-040R-02.pdf

[4] Design and Performance of the OpenBSD Stateful Packet Filter (pf), Daniel Hartmeier, http://www.benzedrine.cx/pf-paper.html, Section 5: Conclusions

introduced by firewalls operating inline between network segments. Rule sets used to enforce security policy can be easily replicated across parallel firewalls, but replication of the in memory connection state table is required to avoid service interruptions to existing connections during a firewall failure. Also required to provide service transition to an alternate firewall is IP and MAC address failover, allowing virtually uninterrupted network traffic flow. This paper will discuss the technology and techniques that can be used to overcome these complications and design a highly available firewall service that can meet the following stringent business requirements.

2.2. Business Requirements

Authorized services that traverse the network must not be excessively impacted by a firewall failure. The detailed business requirements can be translated to a number of network services that need to be protected from firewall server failure. Note these services themselves may not be configured to be highly available, only their transit through the firewall is to be covered in this paper.

The following table describes the services to be covered by this solution. External access to the DMZ, internal access to the DMZ, and internal access to external networks need to be covered by this solution.

Connection Oriented Protocols (TCP based)	**Connectionless Oriented Protocols (UDP, ICMP)**
HTTP HTTPS SMTP IMAP IMAPS SSH DNS syslog Various custom TCP based services	syslog DNS icmp type 8 (echo) icmp type 0 (echo response)

Out of Scope Functions	**Comments**
VPN Tunnels/Protocols	Application protocol level support required
FTP Protocol	Application level proxy required, insecure protocol
Dynamic Port based Applications (ICQ, IM)	Application level proxy required

Table 1 - Business Requirements - Protocols

2.3. Target Service Levels

For all included services, the solution must have the following attributes:

- Allow the use of up to 255 virtual IP addresses
- Employ less than 5 second failover with no established connection loss
- Scheduled or unscheduled maintenance must not incur any service outage
- Maximum window of exposure for rule set synchronization is 15 minutes
- Maximum sustained traffic throughput required is 1.5Mbps Internet traffic and 50Mbps internal to DMZ traffic
- All firewall policy (rule set) management must be through a GUI on a remote administration workstation
- Employ low cost hardware (x86 or SPARC)
- Employ open source software

3. Designing the Solution

The high availability design considered in this paper only focuses on the firewall, although extending this scope to the perimeter Internet and extranet feeds, edge connection devices, and core networks, is a logical next phase. Employing multiple firewalls as a clustered group is possible with the HA software used in this paper, however only two firewalls are shown as our business requirements can be met with a firewall pair.

The firewall solution in this paper is divided into four components:

- Overall Design Assumptions – Restrictions and Design Decisions
- Security Policy Enforcement – Packet inspection and filtering
- High Availability – Clustering of firewall services to withstand failures
- Management – Easy security policy management

Throughout each component, design decisions have been made with reference to industry best practices established by recognized computer network security authorities such as NIST[5], NSA[6], and SANS[7].

3.1. Overall Design Assumptions

Although this paper discusses the integration of open source projects, the reader needs to be familiar with how to build and install OpenBSD software, as well as have a sufficient understanding of network routing and Ethernet traffic flow. Section *3.5 Implementation* gives an outline of issues that need to be addressed and steps required to build the solution presented in this paper.

The network switching fabric was designed to intentionally use hardware separation for the subnets, avoiding the use of VLAN technology for security policy enforcement. Only network zones at the same security level can use VLAN technology and share the same physical switch. Networks with differing security levels must use physically separate switches, to ensure no possibility of employing VLAN hopping techniques[8] to evade security policy.

OpenBSD[9] was selected as the operating system for use on the firewall servers. Not only is OpenBSD a highly secure OS[10], it is also the crucial foundation for the HA components selected for this solution. The man pages within OpenBSD are an excellent source of information, and contain many how-to-use examples. The reader is encouraged to read the man pages to gain a deeper understanding of the technical issues discussed in this paper[11]. The firewall cluster configuration is based on a pair of firewall servers, **fw1** as the primary firewall, and **fw2** as the alternate firewall. Refer to the logical and physical architecture diagrams (Figure 2 and Figure 8) while reading the following design description. It is assumed that the reader has followed the Operating System Security links at the end of this paper to harden the OpenBSD servers used for the firewall services.

The following tables specify the network parameters assigned to each firewall server in the cluster.

	Sync	External	DMZ	Internal

[5] US National Institute of Standards and Technology (www.nist.gov)
[6] US National Security Agency (www.nsa.gov)
[7] System and Network Security organization (www.sans.org)
[8] VLAN Hopping Considerations, General Design Considerations, Windows IT Library, http://www.windowsitlibrary.com/Content/1110/06/2.html
[9] http://www.openbsd.org
[10] http://www.openbsd.org/security.html
[11] Traditional *nix man (manual) pages were not very helpful, normally only providing terse and often circular references. By contrast, the OpenBSD man pages often contain extended descriptions and useful examples.

Hostname	fw1sync	fw1ext	fw1dmz	fw1int
Network Address	10.0.120.0/24	1.2.3.160 / 27	192.168.1.0 / 24	192.168.2.0 / 24
Physical Interface	xl3	xl0	xl1	xl2
Physical IP Address	.253	.175	.253	.253

Table 2 - fw1 Interface Assignments

	Sync	External	DMZ	Internal
Hostname	fw2sync	fw2ext	fw2dmz	fw2int
Network Address	10.0.120.0/24	1.2.3.160 / 27	192.168.1.0 / 24	192.168.2.0 / 24
Physical Interface	xl3	xl0	xl1	xl2
Physical IP Address	.252	.176	.252	.252

Table 3 - fw2 Interface Assignments

Refer to *Figure 8 – Logical Network Diagram* for an illustration of the final firewall server pair network configuration.

3.2. Security Policy Enforcement

Features that this component must provide include packet inspection and filtering on all network interfaces on the firewall servers, according to the installed security policy or rule set. Whenever the rule set specifies, this component must also provide full logging capabilities.

The software selected to provide this component is the OpenBSD Packet Filter (pf)[12], which provides stateful filtering of common IP traffic, such as TCP, UDP, ICMP, as well as Network Address Translation (NAT). pf is an excellent choice for this component, as it has many advanced features to normalize and condition TCP/IP traffic, such as resolving overlapping TCP segments and packet de-fragmentation prior to rule set processing. pf also provides bandwidth control and packet prioritization or queue control features, although they are not explored in this paper.

In this paper, only the highlights of pf will be covered, as the focus is mainly the HA components of this solution, and the security policy will be managed exclusively through the GUI management interface. The reader is encouraged to obtain and read the pf documentation for more in-depth coverage of this software[13]. The features of pf that are used in this solution include stateful packet filtering, using the `keep state` keywords, network address translation (NAT) to map internal private addresses to public internet routable addresses, packet normalization to drop packets with incorrect flag usage and defragment packets before processing with the rule set, as well as extensive packet traffic logging. No application proxies are configured to assist in securely handling complex protocols such as FTP, although these capabilities exist within pf. Also no VPN configuration is addressed in this paper, although pf can handle virtually any IP protocol.

The operation of pf is controlled by a configuration file, typically located at `/etc/pf.conf`, that contains both pf directives and filtering rules. In the configuration of this solution, all pf configuration is done

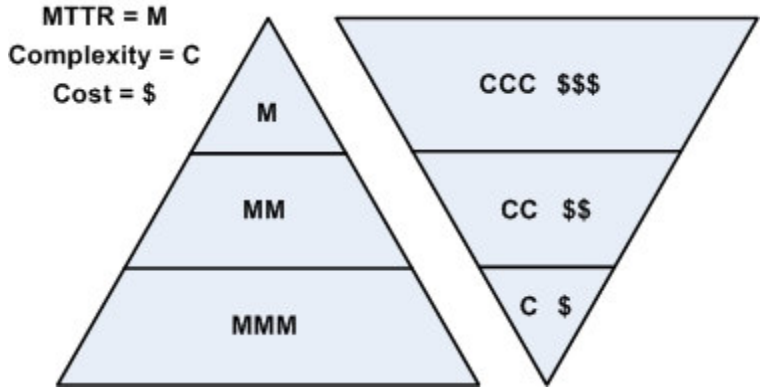
[12] ftp://ftp.openbsd.org/pub/OpenBSD/doc/pf-faq.pdf
[13] http://www.openbsd.org/faq/pf/

through the FWBuilder GUI interface, with the resulting `pf.conf` file transferred to the target firewall via an scp transfer.

Refer to *Figure 2 - Firewall Architecture* for an illustration of where the pf component is employed in the final solution.

3.3. High Availability

The high availability (HA) component of the solution is the most complex due to several advanced feature requirements. Continued service availability in the case of firewall server failure can be handled in several ways, each with progressively shorter mean-time-to-repair (MTTR) cycles. These scenarios can be depicted in an inverted pyramid illustration:



Typical MTTR	Service	Complexity	Typical Cost[14]	Technique
Seconds	Uninterrupted	Very High	Very High	Clustering
< 15 Minutes	Interrupted	High	High	Automated Scripts
2-4 Hours	Interrupted	Moderate	Moderate	Manual Swap using Cold Spares
1-2 Days	Interrupted	Low	Low	Vendor SLA

Due to the increase in functionality when layering HA features onto a solution, the complexity also increases. Referring to stability of network systems, "Complexity is the enemy."[15], as stated by David Willis in Packet Magazine. To avoid the pitfalls of complexity, we employ a solution that is well designed, based on features that will interoperate well to attain the level of HA we need. Also to alleviate the dangers of complexity will be automation – as many of the failover processes will be automated as possible.

Using the resiliency features of TCP, which will retransmit small numbers of lost packets, a solution can be built that offers the perception of uninterrupted service through the firewalls. For the purposes of this paper, the assumption is made that other connectionless protocols will handle lost packets at the application layer, such as DNS query retries. With the resiliency features of the protocols we are passing through the firewalls, and a failover solution that will resume service in a short enough time frame to avoid application connectivity timeouts, we can provide the user with the perception of nearly 100% availability. Testing with the final solution involved establishing stream based service connections through the firewall pair, and invoking various forms of failure, including physically disconnecting interfaces and firewall server reboots. All established and new connections continued operating as required, with only a small (less than 5 second) lag noticeable to the end user.

[14] Commercial magnitude of order costs, based on author's industry experience
[15] David Willis, Chief of Communications Research, Gartner, Cisco Packet Magazine, First Quarter 2006, pg 30

3.3.1. IP and MAC Address Failover

IP and Ethernet MAC address failover is the foundation of the HA services. In the case of a failure of the primary firewall server or one of its interfaces, the well known IP addresses in use on the firewall must move to an alternate firewall server. Although gratuitous address resolution protocol (ARP) broadcasts will allow the use of a new MAC address, the solution used in this paper also fails over the MAC address to speed the resumption of service. This technique of using floating IP and MAC addresses allows the firewall service to migrate to alternate servers as needed to maintain the target service level.

During the research of solutions, two open source implementations of redundancy protocols were found that could handle the management of IP and MAC addresses. The heartbeat[16] project uses the Cisco Virtual Router Redundancy Protocol (VRRP) to advertise and elect a primary among several physical candidates. While this product was tested and found to be an acceptable solution, there exists a possibility of patent infringement as Cisco maintains a patent on the VRRP protocol. Although Cisco has publicly committed to allow free use of VRRP, the caveat they place on its use prompted a search for an alternative. The developers of OpenBSD have created a public domain equivalent of VRRP called Common Address Redundancy Protocol (CARP)[17]. CARP solved some of the technical problems with VRRP and also mitigated the possibility of future patent infringement faced by product developers using VRRP[18].

Readers who are familiar with VRRP will find the following significant advantages of the CARP protocol implementation[19]:

- Address family independent, supporting both IPv4 and IPv6 as both the transport for CARP traffic as well as the floating addresses
- CARP has an "arpbalance" feature which allows multiple hosts to share a single IP address concurrently. This feature can be used to provide a form of load balancing and service resiliency on servers behind a firewall. This feature is not employed in this paper.
- Cryptographically strong SHA-1 HMAC to protect each CARP advertisement

As there are multiple firewall servers to provide service should a component of the primary firewall fail, this paper does not address network interface redundancy[20], although would be required when expanding the HA scope past the firewall servers to encompass the network switching fabric connecting to the firewalls. In this design, switches and routers used to connect the firewalls to the various network segments are still single points of failure.

3.3.2. Configuration of CARP

CARP functionality is employed by establishing CARP interfaces. Similar to bringing up Ethernet interfaces on OpenBSD, CARP interfaces are configured either through a `/etc/hostname.carp`***x*** file or the `ifconfig` command.

From the CARP man page:

> A carp interface can be created at runtime using the `ifconfig carpN` create command or by setting up a `hostname.if`(5) configuration file for `netstart`(8).

[16] http://linux-ha.org/HomePage
[17] http://www.openbsd.org/faq/faq6.html#CARP
[18] Reference to VRRP patent infringement danger, http://www.openbsd.org/faq/faq6.html#CARP
[19] Paraphrased from "Firewall Failover with pfsync and CARP", Ryan McBride, http://www.countersiege.com/doc/pfsync-carp/, pg 2
[20] Refer to the appendix Interface Redundancy for information on changes that would be required to introduce network fabric resiliency.

To use CARP, the administrator needs to configure at minimum a common virtual host ID and virtual host IP address on each machine which is to take part in the virtual group. Additional parameters can also be set on a per-interface basis: `advbase` and `advskew`, which are used to control how frequently the host sends advertisements when it is the master for a virtual host, and `pass` which is used to authenticate CARP advertisements.

These configurations can be done using `ifconfig`(8), or through the SIOCSVH ioctl.

It is important to note that starting in OpenBSD 3.5, the CARP implementation includes the logic to automatically increase the `advskew` for any node which has suffered link loss on any network interface. The effect of this is to decrease the advertisement frequency of a node with a failed network interface, thereby forcing the node to lose the next CARP election and give up its primary role. This election process can complete in as little as 3 seconds, providing a fast failover of an entire firewall in the case that only one network interface fails.

Configuration of failover behavior with CARP allows two methods of operation, either with or without pre-emption. The CARP protocol primary server broadcasts CARP advertisement messages via multicast using IP protocol number 112 (CARP). The alternate servers listen for this broadcast, and if it stops, the alternate servers will begin broadcasting advertisements. The server which advertises the most frequently will most likely be elected the new primary. The frequency of CARP advertisements can be controlled through the use of the `advbase` and `advskew` parameters.

Preemption can be enabled to make a specific firewall server the primary firewall whenever it is operating normally. This is configured in the design discussed in this paper, although there are published opinions which subscribe to the philosophy that when a failover has occurred, the new primary server should remain the primary until it fails. This results in a primary server transition only on primary failure, and in the opinion of the author, the primary server should be a single designated server to allow for easier monitoring and reporting. Preemption mode can be enabled through the following sysctl setting on all firewall servers:

```
/etc/sysctl.conf:  net.inet.carp.preempt=1
```

With preemption enabled, the server which is intended to be the primary needs to have the lowest `advskew` value, and is typically 0. The primary server (fw1) has a CARP `advskew` value of 0, while the alternate server (fw2) has an `advskew` value of 150.

Refer to Figure 2 - Firewall Architecture for an illustration of where the CARP component is employed in the final solution. Refer to *Listing 1* for fw1 CARP configuration and *Listing 2* for fw2 CARP configuration.

	CARP IP	**vhid**	**advskew**	**passwd**
External Network – VLAN2 (1.2.3.160 / 27) Network 1.2.3.160 Broadcast 1.2.3.191				
carp101	.180	101	fw1=0 fw2=150	`s2948f2d8d`
carp102	.181	102	fw1=0 fw2=150	`j20sj3us40`
carp103	.182	103	fw1=0 fw2=150	`2-s89f782a`
carp104	.183	104	fw1=0 fw2=150	`uiS923-aDj`
DMZ Network – VLAN3 (192.168.1.0/24) Network 192.168.1.0 Broadcast 192.168.1.255				
carp111	.1	111	fw1=0 fw2=150	`kLSU710A8f`
carp112	.254	112	fw1=0 fw2=150	`XJdu8a67s0`
Internal Network – VLAN4 (192.168.2.0/24) Network 192.168.2.0 Broadcast 192.168.2.255				
carp121	.1	121	fw1=0 fw2=150	`Kmm7a-f-s8`
carp122	.254	122	fw1=0 fw2=150	`Jhi-8aF8SD`

Table 4 - CARP Interface Definitions (`/etc/hostname.carpx`)

As shown in *Figure 2 - Firewall Architecture*, the pf policy enforcement component protects all of the network interfaces on the firewall. Due to this filtering, although CARP uses cryptographic hashing to protect the CARP advertisements, specific pf rules must be installed to allow the CARP broadcast traffic to pass. This improvement in security is especially necessary on the external and DMZ interfaces, to ensure co-existing or malicious CARP broadcasts are ignored.

Figure 1 - CARP and pfsync Filter Rules, show an example of the pf rule to allow CARP traffic through on the dedicated pfsync interface. Similar rules are required for all of the interfaces that CARP traffic will be encountered on. Generic raw pf rules to allow CARP traffic would look like:

```
pass on { xl0 xl1 xl2 xl3 } proto carp keep state
```

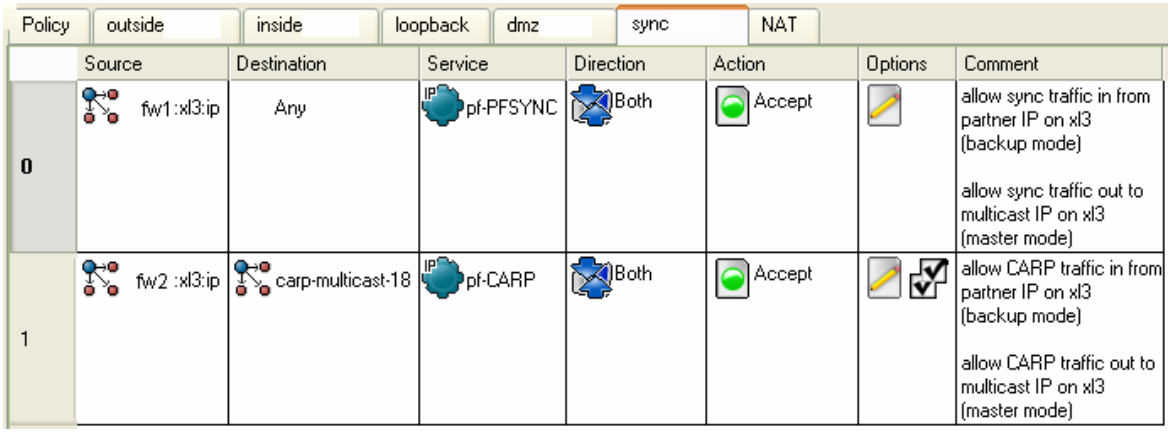


Figure 1 - CARP and pfsync Filter Rules

3.3.3. pfsync for Connection State Table Synchronization

Employing pf to protect the network interfaces on each firewall and CARP to provide resilient IP/MAC address transition between firewall servers, results in a configuration that will have a very short (typically less than 5 seconds) MTTR. This configuration will address all but one of the original business requirements – *uninterrupted existing connections*. With the current configuration, in the case of a firewall server failure, the alternate firewall will promote itself to be the primary firewall within the desired service level. All new connections established through the new primary firewall will function as expected as long as the pf rule sets are synchronized on both firewall servers. Existing connections will still be dropped, as the new primary firewall only sees them as in-progress connections that it does not have listed in its in memory connection state table.

pfsync is a component of OpenBSD which provides near real-time synchronization of the connection state table, preserving established connections in the case of a primary firewall failure. pfsync performs a multicast broadcast using IP protocol 240 (pfsync) to transfer state insertion, deletion, and update activity to all participating alternate firewall servers. The receiving firewall servers listen for these messages and dynamically modify their connection state tables to allow uninterrupted operation of established connections in the case any of the receiving servers are promoted to be the primary firewall.

As there is no significant security employed within the pfsync protocol, a dedicated layer 2 connection must be provided for secure connection state table updates. The lack of authentication and privacy features in the pfsync protocol was initially to provide for very low latency in transmission of state table updates. With the firewall pair shown in this solution, an Ethernet cross-over cable was used to connect the `xl3` interface on each server.

Similar to regular Ethernet and CARP interfaces, the pfsync mechanism is brought up by configuring the pfsync interface, with a configuration file like:

```
/etc/hostname.pfsync0:   up syncif xl3
```

Figure 1 - CARP and pfsync Filter Rules shows the rules used to allow pfsync protocol traffic only on the dedicated pfsync interfaces (xl3). Generic raw pf rules to allow pfsync traffic would look like:

```
pass quick on { xl3 } proto pfsync
```

Note that the pf key words "keep state" are not used for pfsync traffic, as it is connectionless broadcast traffic. *Listing 1 - fw1 CARP configuration* and *Listing 2 - fw2 CARP configuration* show the CARP interface definition files (`/etc/hostname.carp`***x***).

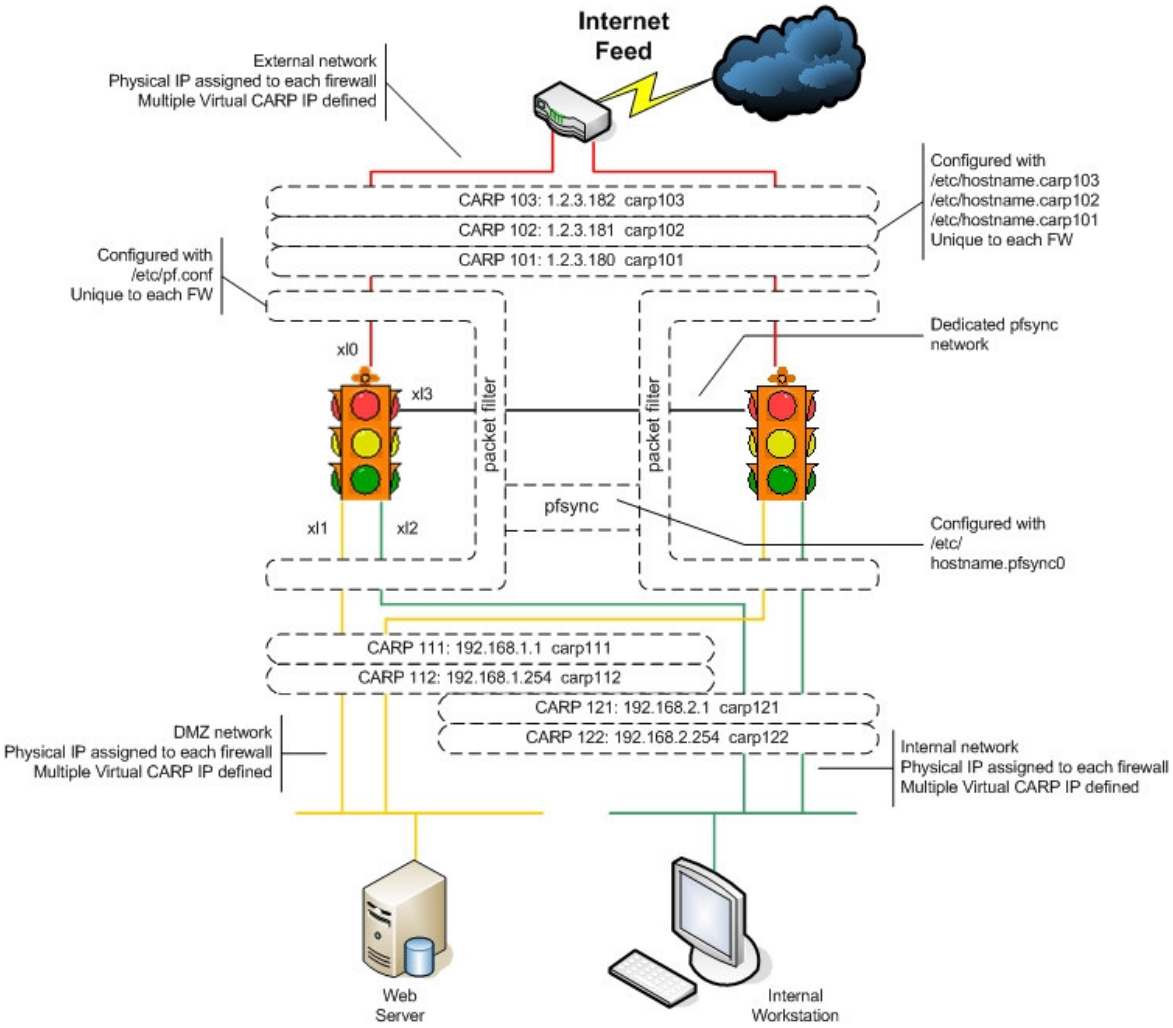


Figure 2 - Firewall Architecture

CARP configuration files for all floating IP addresses are shown below and reside in /etc on each firewall server. Note that since this solution is using pre-emption, the configuration files differ between the primary and alternate firewall. Elements in the configuration file (left to right):

- protocol family (IPv4)
- Virtual IP address
- Subnet mask

- Broadcast address
- Virtual CARP ID keyword and number
- CARP Advertisement skew keyword and value (Alternate server only)
- CARP Password keyword and value

```
hostname.carp101
inet 1.2.3.180 255.255.255.224 1.2.3.191 vhid 101 pass s2948f2d8d

hostname.carp102
inet 1.2.3.181 255.255.255.224 1.2.3.191 vhid 102 pass j20sj3us40

hostname.carp103
inet 1.2.3.182 255.255.255.224 1.2.3.191 vhid 103 pass 2-s89f782a

hostname.carp104
inet 1.2.3.183 255.255.255.224 1.2.3.191 vhid 104 pass uiS923-aDj

hostname.carp111
inet 192.168.1.1 255.255.255.0 192.168.1.255 vhid 111 pass kLSU710A8f

hostname.carp112
inet 192.168.1.254 255.255.255.0 192.168.1.255 vhid 112 pass XJdu8a67s0

hostname.carp121
inet 192.168.2.1 255.255.255.0 192.168.2.255 vhid 121 pass Kmm7a-f-s8

hostname.carp122
inet 192.168.2.254 255.255.255.0 192.168.2.255 vhid 122 pass Jhi-8aF8SD
```

Listing 1 - fw1 CARP configuration

```
hostname.carp101
inet 1.2.3.180 255.255.255.224 1.2.3.191 vhid 101 advskew 150 pass s2948f2d8d

hostname.carp102
inet 1.2.3.181 255.255.255.224 1.2.3.191 vhid 102 advskew 150 pass j20sj3us40

hostname.carp103
inet 1.2.3.182 255.255.255.224 1.2.3.191 vhid 103 advskew 150 pass 2-s89f782a

hostname.carp104
inet 1.2.3.183 255.255.255.224 1.2.3.191 vhid 104 advskew 150 pass uiS923-aDj

hostname.carp111
inet 192.168.1.1 255.255.255.0 192.168.1.255 vhid 111 advskew 150 pass kLSU710A8f

hostname.carp112
inet 192.168.1.254 255.255.255.0 192.168.1.255 vhid 112 advskew 150 pass XJdu8a67s0

hostname.carp121
inet 192.168.2.1 255.255.255.0 192.168.2.255 vhid 121 advskew 150 pass Kmm7a-f-s8

hostname.carp122
```

```
inet 192.168.2.254 255.255.255.0 192.168.2.255 vhid 122 advskew 150 pass Jhi-8aF8SD
```

Listing 2 - fw2 CARP configuration

3.4. Management

To simplify writing and management of the firewall Security Policy, a Graphical User Interface (GUI) will be used on an administrator workstation. Firewall Builder[21] (FWBuilder) is a multi-firewall configuration and management interface that allows administrators to apply a common policy to disparate underlaying firewall technology. Although in this paper the administrator workstation is configured as a desktop running Microsoft Windows, FWBuilder is supported on most major distributions of Linux and BSD as well.

FWBuilder is comprised of a GUI to build an abstract firewall rule set, an object oriented rule and entity store, and a variety of compilers to convert the vendor neutral rule set into a rule set suitable for the target firewall platform. Policy editing is simple and intuitive with the use of drag and drop operations. The FWBuilder project maintains policy compilers for a variety of firewall platforms including iptables, IP Filter, OpenBSD pf, and Cisco PIX. Employing these underlaying policy compilers allow the application of the same rule set to differing firewall platforms, allowing the ability to centralize policy management, and ease migration to different platforms.

From the FWBuilder web site:

> Firewall Builder allows for management of multiple firewalls using the same network object database. Change made to an object is immediately reflected in the policy of all firewalls using this object. Administrator only needs to recompile and install policies on actual firewall machines.
>
> In Firewall Builder, administrator works with an abstraction of firewall policy and NAT rules; software effectively "hides" specifics of particular target firewall platform and helps administrator focus on implementation of security policy. Backend software components, or policy compilers, can deduct many parameters of policy rules using information available through network and service objects and therefore generate fairly complex code for the target firewall, thus relieving administrator from having to remember all its details and limitations. Policy compilers can also run sanity checks on firewall rules and make sure typical errors are caught before generated policy is deployed.

The packet filtering rule set used is shown in *Figure 3 - FWBuilder GUI Interface*, while the NAT and PAT translation is shown in *Figure 5 - Firewall NAT/PAT Rules*. Both the Policy and NAT tabs within FWBuilder specify global policy. These rules are not constrained by any specific network interface, but operate after evaluation of rules specific to each interface have been evaluated. FWBuilder also provides the ability to specify rules which operate on individual physical interfaces. These tabs are typically used only for anti-spoofing or fine grained global policy exceptions, such as allowing CARP or pfsync traffic – as shown in *Figure 4 - Interface Policy*.

[21] http://www.fwbuilder.org

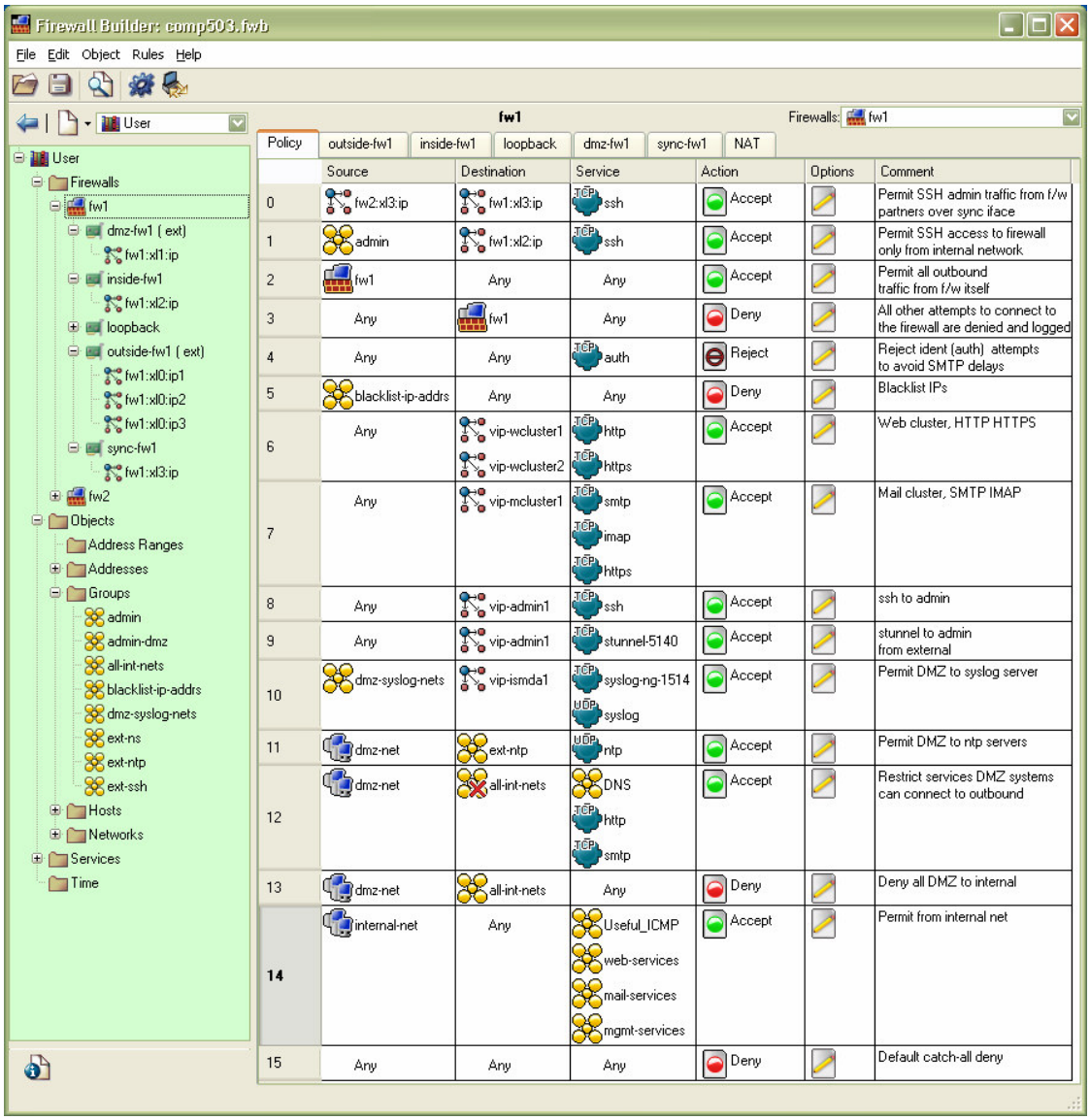


Figure 3 - FWBuilder GUI Interface

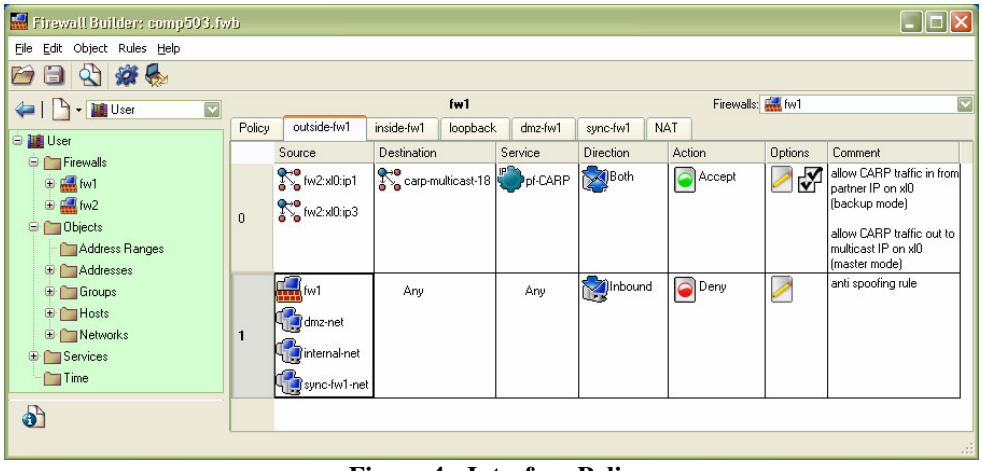


Figure 4 - Interface Policy

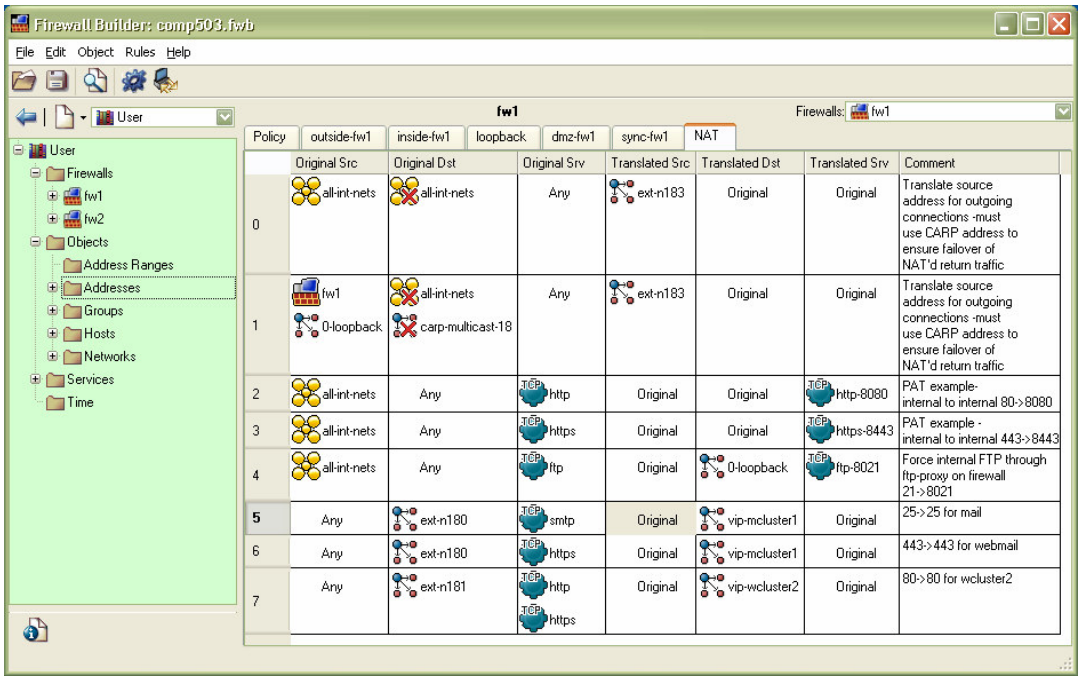


Figure 5 - Firewall NAT/PAT Rules

Configuration of the OpenBSD pf parameters is done through drilling into the desired firewall object and selecting each of the configuration tabs:

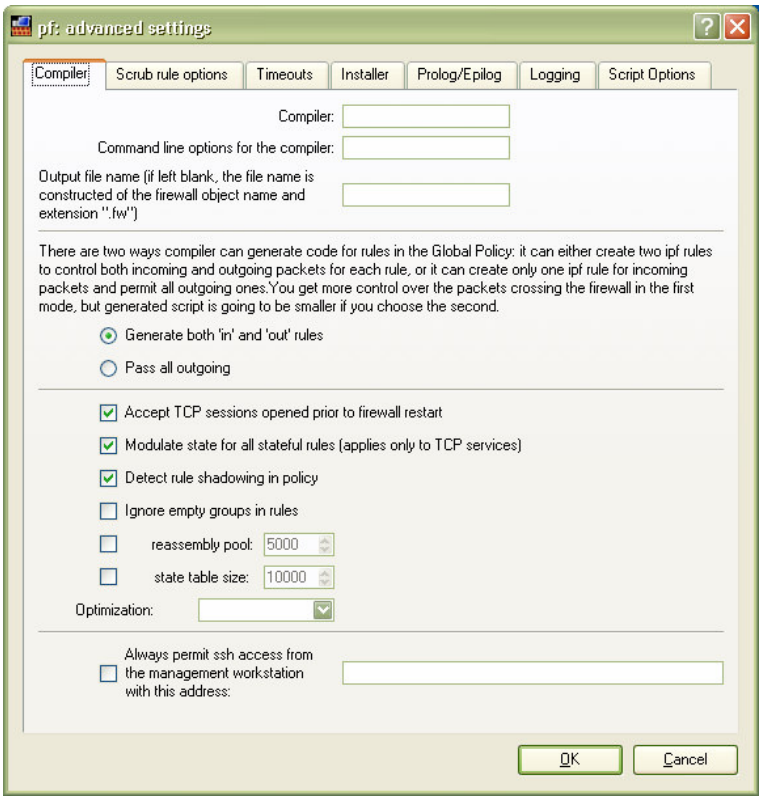


Figure 6 - Firewall Configuration in FWBuilder

3.5. Implementation

3.5.1. Hardware Used

The hardware that was selected for the new firewall servers consists of two Compaq AP550 workstations with 1GHz Intel CPU, 256MB memory, 9GB hard disk, and 4x 3C905 100Mbps Ethernet NICs. Each firewall machine is located in a separate rack to take advantage of the independent 20A 1 hour UPS units in each rack. With each rack having dedicated switches for each security domain and a dedicated UPS, there is no single point of failure within the servers across both racks. With a moderate policy size of 25 global rules, 25 global NAT rules, and less than 10 individual interface rules, traffic to the Internet feed can saturate the link and still have almost 75% CPU capacity remaining.

Installing the servers across two racks quickly revealed a potential management challenge with cable management. To avoid confusion during future upgrade work, network security zones were assigned colors as follows:

Network Security Zone	Cable Colour
External Networks	Red
DMZ Networks	Yellow
Internal Networks	Green
Synchronization Networks	White
Trunk Cables between Racks	Orange

This is illustrated in *Figure 7 - Firewall Cables and Cable Harness*.

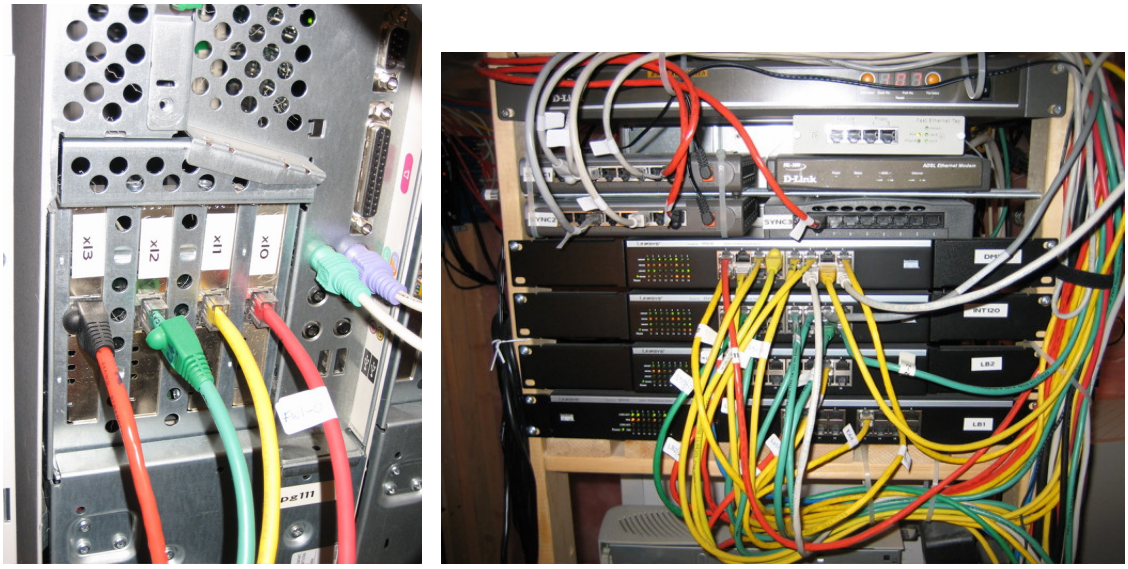


Figure 7 - Firewall Cables and Cable Harness

The final network architecture is illustrated in *Figure 8 – Logical Network Diagram*.

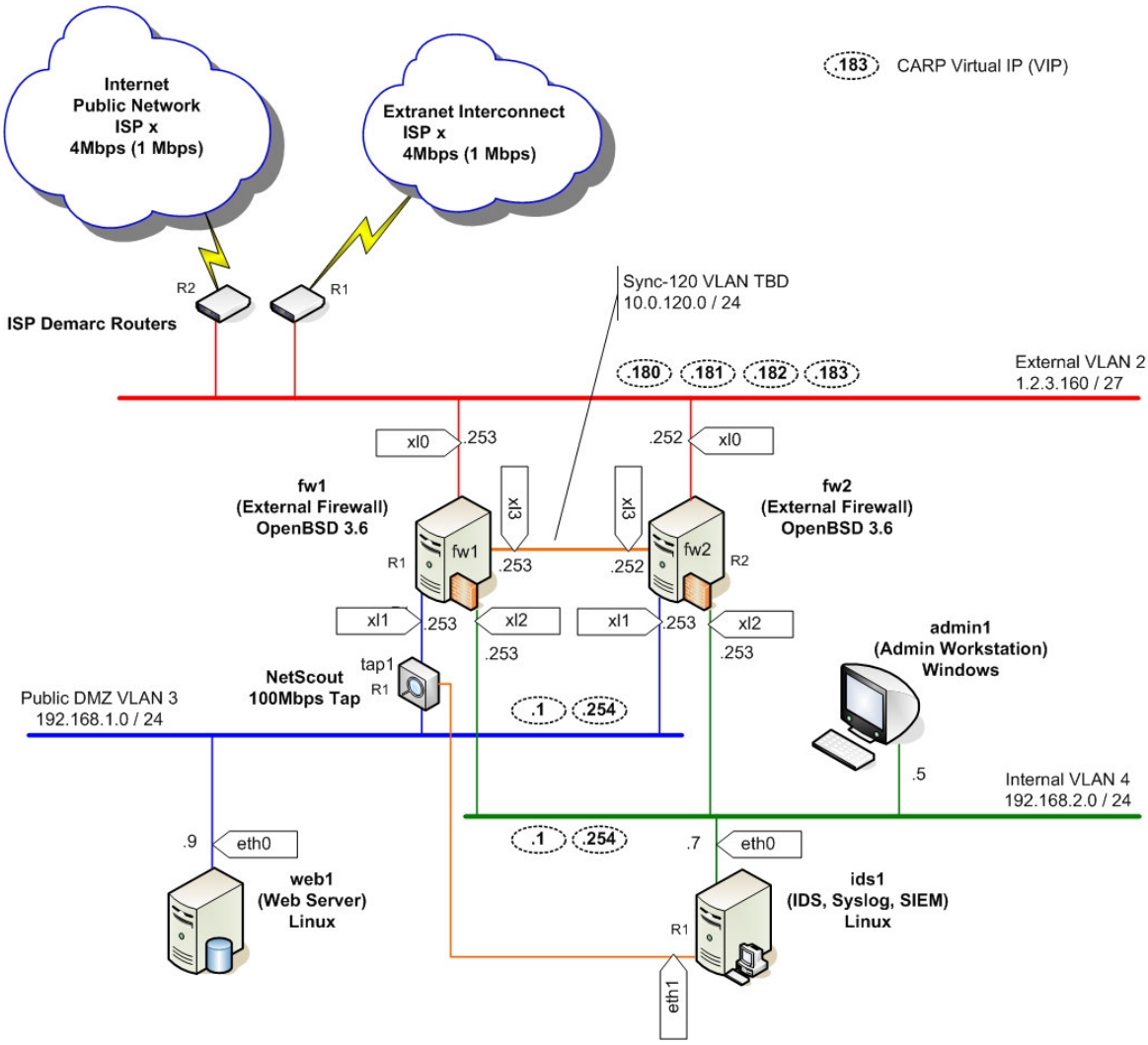


Figure 8 – Logical Network Diagram

3.5.2. Server Configuration

All initial installation and configuration of the hardware and the software was done in complete physical isolation from any networks, including both internal and especially external networks. Only after the operating system was fully configured, including the following base firewall rule set, which allows only administrative SSH access, were the firewalls connected to the internal networks for testing. After vulnerability testing using nmap[22] and Nessus[23] security tools revealed no weaknesses, the firewalls were attached to the external networks as well.

The OpenBSD operating system was installed on both firewalls, with the same configuration. Most of the installation defaults were taken, and only minimal software (base) was loaded to improve security.

Post Installation Customization

[22] http://www.insecure.org/nmap/
[23] http://www.nessus.org/

Activation of pf on boot is enabled by adding

```
pf=YES
```

to the configuration file `/etc/rc.conf.local`. Any system configuration changes should be done using this file, which is preserved across system upgrades.

CARP pre-emption is enabled by adding

```
net.inet.carp.preempt=1
```

to the system configuration file `/etc/sysctl.conf`.

Activation of network interfaces is enabled through /etc/hostname.if files, which give OpenBSD enough information to create a network device and bring up the interface. *Listing 1* and *Listing 2* show the hostname files for each CARP interface. After each firewall has the physical and CARP interfaces defined, the network interfaces will look like what is shown in *Listing 3* and *Listing 4*.

Abbreviated listings[24]:

Listing 3 - Primary firewall interface state (fw1)

```
fw1# ifconfig -a
xl0: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
        address: 00:50:da:64:46:1f
        media: Ethernet autoselect (10baseT)
        status: active
        inet 1.2.3.175 netmask 0xfffffffe0 broadcast 1.2.3.191
xl1: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
        address: 00:50:da:1f:b5:98
        media: Ethernet 100baseTX full-duplex
        status: active
        inet 192.168.1.253 netmask 0xffffff00 broadcast 192.168.1.255
xl2: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
        address: 00:50:04:b0:a6:56
        media: Ethernet 100baseTX full-duplex
        status: active
        inet 192.168.2.253 netmask 0xffffff00 broadcast 192.168.2.255
xl3: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        address: 00:50:da:12:cc:3b
        media: Ethernet autoselect (100baseTX full-duplex)
        status: active
        inet 10.0.120.253 netmask 0xffffff00 broadcast 10.0.120.255

pflog0: flags=141<UP,RUNNING,PROMISC> mtu 33224
pfsync0: flags=41<UP,RUNNING> mtu 1348
        pfsync: syncif: xl3 syncpeer: 224.0.0.240 maxupd: 128

carp101: flags=41<UP,RUNNING> mtu 1500
        carp: MASTER vhid 101 advbase 1 advskew 0
        inet 1.2.3.180 netmask 0xfffffffe0
carp102: flags=41<UP,RUNNING> mtu 1500
        carp: MASTER vhid 102 advbase 1 advskew 0
        inet 1.2.3.181 netmask 0xfffffffe0
carp103: flags=41<UP,RUNNING> mtu 1500
        carp: MASTER vhid 103 advbase 1 advskew 0
```

[24] Command output has been trimmed to conserve page space

```
        inet 1.2.3.182 netmask 0xfffffffe0
carp104: flags=41<UP,RUNNING> mtu 1500
        carp: MASTER vhid 104 advbase 1 advskew 0
        inet 1.2.3.183 netmask 0xfffffffe0

carp111: flags=41<UP,RUNNING> mtu 1500
        carp: MASTER vhid 111 advbase 1 advskew 0
        inet 192.168.1.1 netmask 0xffffff00
carp112: flags=41<UP,RUNNING> mtu 1500
        carp: MASTER vhid 112 advbase 1 advskew 0
        inet 192.168.1.254 netmask 0xffffff00

carp121: flags=41<UP,RUNNING> mtu 1500
        carp: MASTER vhid 121 advbase 1 advskew 0
        inet 192.168.2.1 netmask 0xffffff00
carp122: flags=41<UP,RUNNING> mtu 1500
        carp: MASTER vhid 122 advbase 1 advskew 0
        inet 192.168.2.254 netmask 0xffffff00
fw1#
```

Listing 4 - Backup firewall interface state (fw2)

```
fw2# ifconfig -a
xl0: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
        address: 00:50:04:d1:41:9c
        media: Ethernet autoselect (10baseT)
        status: active
        inet 1.2.3.176 netmask 0xfffffffe0 broadcast 1.2.3.191
xl1: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
        address: 00:50:da:2c:0a:c0
        media: Ethernet autoselect (100baseTX full-duplex)
        status: active
        inet 192.168.1.252 netmask 0xffffff00 broadcast 192.168.1.255
xl2: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
        address: 00:50:da:1f:a8:d0
        media: Ethernet autoselect (100baseTX full-duplex)
        status: active
        inet 192.168.2.252 netmask 0xffffff00 broadcast 192.168.2.255
xl3: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        address: 00:50:da:12:cc:3b
        media: Ethernet autoselect (100baseTX full-duplex)
        status: active
        inet 10.0.120.253 netmask 0xffffff00 broadcast 10.0.120.255

pflog0: flags=141<UP,RUNNING,PROMISC> mtu 33224
pfsync0: flags=41<UP,RUNNING> mtu 1348
        pfsync: syncif: xl3 syncpeer: 224.0.0.240 maxupd: 128

carp101: flags=41<UP,RUNNING> mtu 1500
        carp: BACKUP vhid 101 advbase 1 advskew 150
        inet 1.2.3.180 netmask 0xfffffffe0
carp102: flags=41<UP,RUNNING> mtu 1500
        carp: BACKUP vhid 102 advbase 1 advskew 150
```

```
        inet 1.2.3.181 netmask 0xfffffffe0
carp103: flags=41<UP,RUNNING> mtu 1500
        carp: BACKUP vhid 103 advbase 1 advskew 150
        inet 1.2.3.182 netmask 0xfffffffe0
carp104: flags=41<UP,RUNNING> mtu 1500
        carp: BACKUP vhid 104 advbase 1 advskew 150
        inet 1.2.3.183 netmask 0xfffffffe0

carp111: flags=41<UP,RUNNING> mtu 1500
        carp: BACKUP vhid 111 advbase 1 advskew 150
        inet 192.168.1.1 netmask 0xffffff00
carp112: flags=41<UP,RUNNING> mtu 1500
        carp: BACKUP vhid 112 advbase 1 advskew 150
        inet 192.168.1.254 netmask 0xffffff00

carp121: flags=41<UP,RUNNING> mtu 1500
        carp: BACKUP vhid 121 advbase 1 advskew 150
        inet 192.168.2.1 netmask 0xffffff00
carp122: flags=41<UP,RUNNING> mtu 1500
        carp: BACKUP vhid 122 advbase 1 advskew 150
        inet 192.168.2.254 netmask 0xffffff00
fw2#
```

3.5.3. Packet Filter Logging

Firewalls logs are a crucial part of security posture monitoring. Logging by pf is performed by a daemon `pflogd`, which reads the pflog0 interface and writes packet data to a binary tcpdump format log file /var/log/pflog. Optional logging keywords are used on pf rules to enable capture of packet information when a rule is triggered. pf rules need to specify either block or pass action to trigger a packet log entry. On rules that keep state, only the first packet seen related to the connection will trigger a log entry. pf rule log option syntax is shown here[25]:

```
pass in log [all] on $ext_if inet proto tcp to $ext_if port 22 keep state
```

Will log all incoming packets destined to port 22, including all packets in the conversation if the `all` modifier is specified.

Excerpts from the pf logging FAQ:

> **Reading a Log File**
>
> The log file written by pflogd is in binary format and cannot be read using a text editor. tcpdump must be used to view the log.
>
> To view the log file:
>
> ```
> # tcpdump -n -e -ttt -r /var/log/pflog
> ```
>
> Note that using tcpdump(8) to watch the pflog file does not give a real-time display. A real-time display of logged packets is achieved by using the pflog0 interface:
>
> ```
> # tcpdump -n -e -ttt -i pflog0
> ```

[25] Excerpt from OpenBSD pf logging FAQ, http://www.openbsd.org/faq/pf/logging.html

NOTE: When examining the logs, special care should be taken with tcpdump's verbose protocol decoding (activated via the -v command line option). Tcpdump's protocol decoders do not have a perfect security history. At least in theory, a delayed attack could be possible via the partial packet payloads recorded by the logging device. It is recommended practice to move the log files off of the firewall machine before examining them in this way.

Additional care should also be taken to secure access to the logs. By default, pflogd will record 96 bytes of the packet in the log file. Access to the logs could provide partial access to sensitive packet payloads (like telnet(1) or ftp(1) usernames and passwords).

Filtering Log Output

Because pflogd logs in tcpdump binary format, the full range of tcpdump features can be used when reviewing the logs. For example, to only see packets that match a certain port:

```
# tcpdump -n -e -ttt -r /var/log/pflog port 80
```

This can be further refined by limiting the display of packets to a certain host and port combination:

```
# tcpdump -n -e -ttt -r /var/log/pflog port 80 and host 192.168.1.3
```

The same idea can be applied when reading from the pflog0 interface:

```
# tcpdump -n -e -ttt -i pflog0 host 192.168.4.2
```

Note that this has no impact on which packets are logged to the pflogd log file; the above commands only display packets as they are being logged.

In addition to using the standard tcpdump(8) filter rules, OpenBSD's tcpdump filter language has been extended for reading pflogd output, by specifying the following keywords:

ip	- address family is IPv4.
ip6	- address family is IPv6.
on int	- packet passed through the interface int.
ifname int	- same as on int.
ruleset name	- the ruleset/anchor that the packet was matched in.
rulenum num	- the filter rule that the packet matched was rule number num.
action act	- the action taken on the packet. Possible actions are pass and block.
reason res	- the reason that action was taken. Possible reasons are match, bad-offset, fragment, short, normalize, memory, bad-timestamp, congestion, ip-option, proto-cksum, state-mismatch, state-insert, state-limit, src-limit, and synproxy.
inbound	- packet was inbound.
outbound	- packet was outbound.

Example:

```
# tcpdump -n -e -ttt -i pflog0 inbound and action block and on wi0
```

> This display the log, in real-time, of inbound packets that were blocked on the wi0 interface.

Packet Logging Through Syslog

The design used in this paper writes pf log data to local /var/log/pflog, and sends the log data to a central syslog server. The central syslog server is an enterprise security management system or SIEM. This central system accepts security log data from the IDS sensors, operating system logs, vulnerability scans, and other security data sources for aggregation and correlation. The OpenBSD pf configuration needed to be modified to send the pflog data over syslog to the central security server.

As shown in the pf logging FAQ[26] two shell scripts need to be added to the firewalls, and system configuration files need to be altered. The FAQ describes these changes well and is quoted here:

> First we have to create a user, pflogger, with a /sbin/nologin shell. The easiest way to create this user is with `adduser`(8).
>
> After creating the user pflogger, create the following two scripts:
>
> /etc/pflogrotate:
>
> ```
> #!/bin/sh
> FILE=/home/pflogger/pflog5min.$(date "+%Y%m%d%H%M")
> kill -ALRM $(cat /var/run/pflogd.pid)
> if [ $(ls -l /var/log/pflog | cut -d " " -f 8) -gt 24 ]; then
>    mv /var/log/pflog $FILE
>    chown pflogger $FILE
>    kill -HUP $(cat /var/run/pflogd.pid)
> fi
> ```
>
> /home/pflogger/pfl2sysl:
>
> ```
> #!/bin/sh
> # feed rotated pflog file(s) to syslog
> # do not start if another instance of this script is already running
> pgrep pfl2sysl >/dev/null 2>&1
> if [ $? -ne 0 ] ; then
>         for logfile in /home/pflogger/pflog5min* ; do
>                 if [ -f "$logfile" ] ; then
>                         tcpdump -n -e -ttt -r $logfile | logger -t pf -p
> local0.info
>                         rm $logfile
>                 fi
>         done
> fi
> ```
>
> Edit root's cron job:
>
> ```
> # crontab -u root -e
> ```
>
> Add the following two lines:
>
> ```
> # rotate pf log file every 5 minutes
> ```

[26] OpenBSD pf logging FAQ, http://www.openbsd.org/faq/pf/logging.html

```
0-59/5 * * * * /bin/sh /etc/pflogrotate
```

Create a cron job for user pflogger:

```
# crontab -u pflogger -e
```

Add the following two lines:

```
# feed rotated pflog file(s) to syslog
0-59/5 * * * * /bin/sh /home/pflogger/pfl2sysl
```

Add the following line to /etc/syslog.conf:

```
local0.info     /var/log/pflog.txt
```

If you also want to log to a remote log server, add the line:

```
local0.info     @syslogger
```

Make sure host **syslogger** has been defined in the hosts(5) file.

Create the file /var/log/pflog.txt to allow syslog to log to that file.

```
# touch /var/log/pflog.txt
```

Make syslogd notice the changes by restarting it:

```
# kill -HUP $(cat /var/run/syslog.pid)
```

All logged packets are now sent to /var/log/pflog.txt. If the second line is added they are sent to the remote logging host **syslogger** as well.

The script /etc/pflogrotate now processes and then deletes /var/log/pflog so rotation of pflog by newsyslog(8) is no longer necessary and should be disabled. However, /var/log/pflog.txt replaces /var/log/pflog and rotation of it should be activated. Change /etc/newsyslog.conf as follows:

```
#/var/log/pflog        600    3    250    *    ZB /var/run/pflogd.pid
/var/log/pflog.txt     600    7    *      24
```

PF will now log in ASCII to /var/log/pflog.txt. If so configured in /etc/syslog.conf, it will also log to a remote server. The logging is not immediate but it can take up to about 5-6 minutes (the cron job interval) before the logged packets appear in the file.

After configuring both firewalls with the above procedure, the security monitoring tool was receiving packet log data:

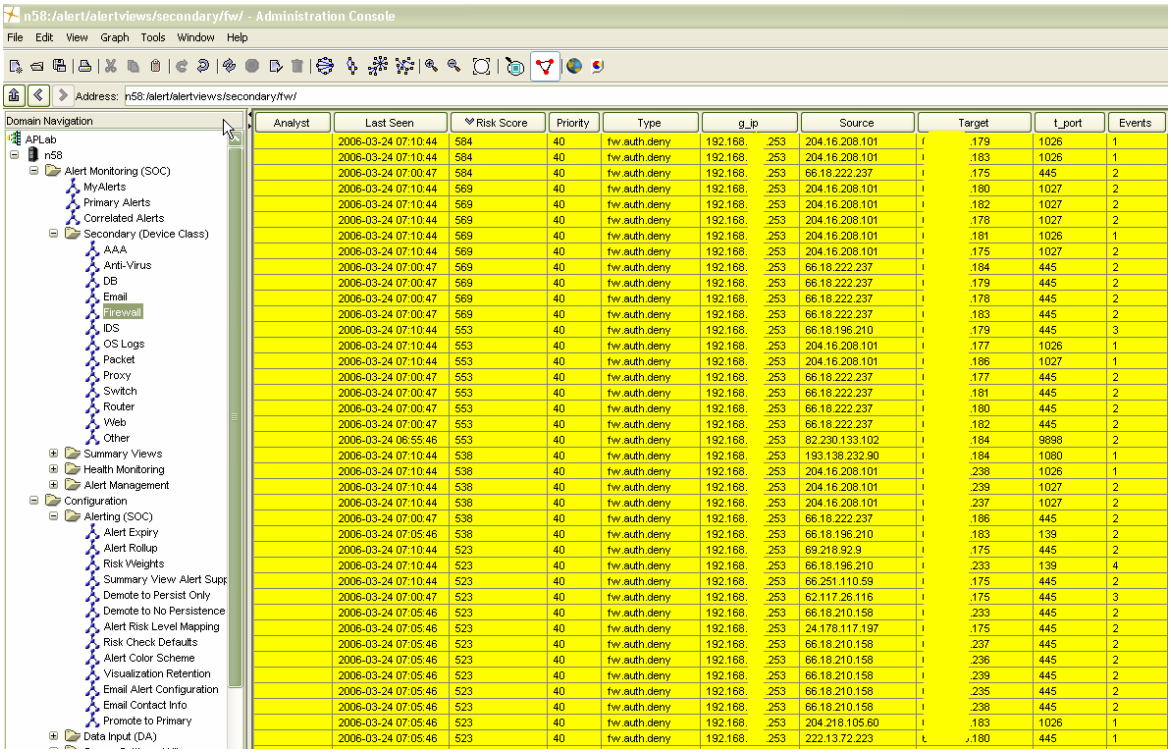


Figure 9 - Security Manager Log Data

4. Solution Benefit Summary

The single largest impact of this solution is providing high quality firewall security services continuously even in the case of a primary firewall failure. This solution has shown that an economical alternative to commercial high availability can be built from open source projects.

An interesting side effect of implementing an HA solution is the ability to take arbitrary firewalls offline without impacting production business functionality. This makes it significantly easier to maintain a regular maintenance and patching schedule – which in turn improves the overall security posture of the organization.

Appendices

A. References

1. Firewall Failover with pfsync and CARP, Ryan McBride, http://www.countersiege.com/doc/pfsync-carp/
2. Packet Filter (pf) FAQ, ftp://ftp.openbsd.org/pub/OpenBSD/doc/pf-faq.pdf
3. High Availability Design, Techniques and Processes, Piedad & Hawkins, Prentice hall, 2001
4. Self-Managed Systems and Services, Jean-Philippe Martin-Flatin, Joe Sventek, and Kurt Geihs, Communications of the ACM, March 2006/Vol. 49, No. 3, pg 37
5. SANS (System and Network Security) Reading Room documents – Firewall and Perimeter Security www.sans.org/rr/
6. Internetworking with TCP/IP Volume III, Douglas Comer, David Stevens, Prentice-Hall, 1993
7. Building Open Source Network Security Tools, Mike Schiffman, Wiley, 2003
8. National Institute of Standards and Technology (NIST) – Special Publications, http://csrc.nist.gov/publications/nistpubs/800-41/sp800-41.pdf, SP800-41 Guidelines on Firewalls and Firewall Policy
9. National Institute of Standards and Technology (NIST) – Special Publications, http://csrc.nist.gov/publications/nistpubs/800-14/800-14.pdf, SP800-14 Generally Accepted Principles and Practices for Securing Information Technology Systems
10. Network Nirvana, Joanna Holmes, Packet Magazine, First Quarter 2006, pg 29

B. Operating System Security

NIST is a US Government agency that publishes best practices documents related to network and computer security.

> The National Institute of Standards and Technology (NIST) is cooperating with other federal agencies, IT vendors, and with industry to advance the development and use of security configuration checklists. A security configuration checklist (sometimes called a security configuration guide, lockdown guide, hardening guide, security technical implementation guide, or benchmark) is basically a series of instructions for configuring an information technology (IT) product to an operational environment. Checklists can be useful tools for reducing vulnerabilities to systems, especially for small organizations with limited resources.
>
> http://csrc.nist.gov/publications/nistpubs/
>
> Special Publication 800-70: Security Configuration Checklists Program for IT Products, http://csrc.nist.gov/checklists/docs/SP_800-70_20050526.pdf

SANS is a vendor neutral industry organization that focuses on Systems and Network Security.

> SANS is the most trusted and by far the largest source for information security training and certification in the world. It also develops, maintains, and makes available at no cost, the largest collection of research documents about various aspects of information security, and it operates the Internet's early warning system - Internet Storm Center. The SANS (SysAdmin, Audit, Network, Security) Institute was established in 1989 as a cooperative research and education organization. Its programs now reach more than 165,000 security professionals, auditors, system administrators, network administrators, chief information security officers, and CIOs who share the lessons they are learning and jointly find

> solutions to the challenges they face. At the heart of SANS are the many security practitioners in government agencies, corporations, and universities around the world who invest hundreds of hours each year in research and teaching to help the entire information security community.

SANS InfoSec Reading Room Documents on Computer and Network Security
http://www.sans.org/rr/

C. Network Interface Redundancy

Ethernet network interface resiliency is required to build switching fabric high availability. Keep in mind that terminology in use differs between some open source projects and the enterprise networking vendors. In many cases, the term **channel bonding**, is referring to active / passive channel failover versus **channel aggregation**. Pairing the Ethernet channel bonding capability inherent in the Linux and some *BSD kernels with multiple switches and host NICs results in a resilient network backbone to build on. If there is a switch failure, the **channel bonding** software will redirect Ethernet frames out the surviving interface with no loss of TCP sessions. Channel aggregation features are beginning to be common in many channel bonding projects, resulting in substantially higher throughputs as multiple Ethernet channels are used in parallel.