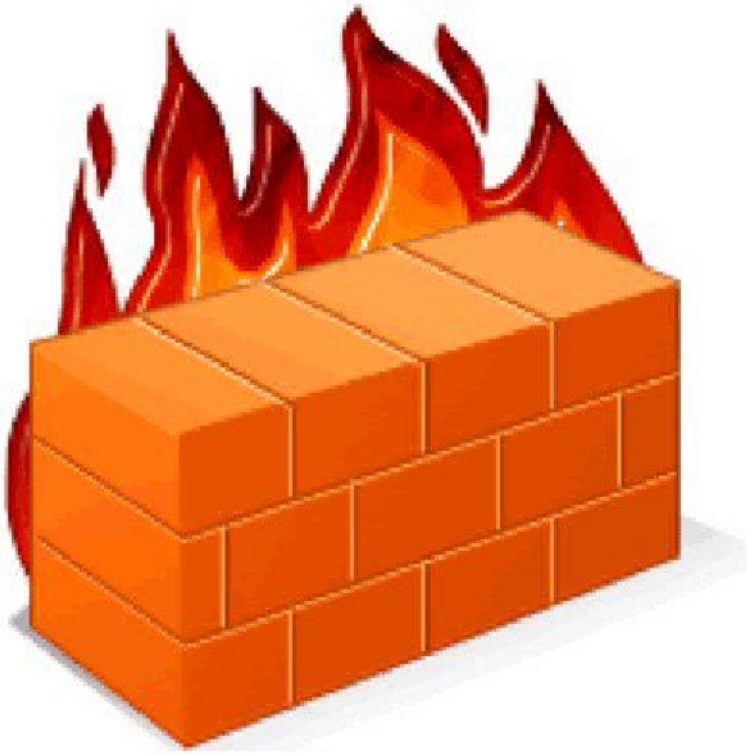
Building a FreeBSD Firewall Appliance

PF, IPFW, ALTQ, DUMMYNET, DHCP, DNS Server,
OpenVPN, Apache, Nagios, WEBMIN and Active Directory Authentication



By: Ian Evans
Version 2

Document Review and Approval

Name	Title	Action	Date
Ian Evans	Author	Initial Draft	11/29/09
Ian Evans	Author	First Production Document	12/12/09
Ian Evans	Author	Added Apache and Nagios	12/26/09

Document History

Version	Title	Date	Author
1	Draft	11/19/09	Ian Evans
2	Production	12/12/09	Ian Evans

Document Contributors and Authors

Title	Name
Author	Ian Evans

Table of Contents

Use this guide at your own risk! Always test in a stage environment prior to production use!

One of the most challenging tasks for system administrators can be finding a firewall that meets all of their needs without breaking the bank. This solution focuses on creating a robust, full-featured firewall that is completely free of charge, granted you have an old system around that you can use to host the software.

PF has been around for many years and is used in some of the best commercial firewall products on the market today. This guide will show you how to build a firewall appliance that is capable of handling traffic for businesses, large and small.

COLOR KEY:

COMMAND
CONFIG FILE CONTENTS
EXTERNAL CONFIGURATION

Recommended hardware:

- I recommend using something like an Athlon 64 or higher for the appliance. You can also use a lower power CPU like an Atom, which will work fine for PF, but may not hold up when additional services are added.
- 4GB of ECC DDR-800 memory is recommended.
- SuperMicro makes a very stable server motherboard. Try to find one of the industrial quality boards that uses solid caps. Nvidia Nforce chipsets seem to work extremely well with BSD (ACPI, Power Management, etc).
- You will need three Gigabit network cards (the WAN port can be 100Mbps). Intel Pro1000's (e1000 driver) are good network cards and are widely supported. I have also used Syskonnect and Nvidia Nforce network cards with great success.
- Hardware RAID is always nice to have in a system you expect reliability from. Hard drives are one of the riskiest components within a system, so protect them with a decent hardware RAID card. I like AMCC (3Ware cards) because they are easy to use, non proprietary and have an excellent web interface.
- Use WD RE3/4 or Seagate ES.2 7200RPM 1.2MTBF SATA drives. Don't go with crap drives... you will regret it later. Do at least a mirrored RAID set to ensure fail-over in case one drive dies. Elaborate RAID (10, 6, 50, etc) is not necessary for this box.

Download and install FreeBSD 7.x.

Download FreeBSD 7.x from http://freebsd.org. I am using FreeBSD 7.2 x64 in this guide.

- Select the standard installation and install the entire ports tree. This makes finding a package easier at a later time.
- Select the default partitioning options. This can be accomplished by selecting "A" for auto and then "Q" to save. FreeBSD fixed the limitation with the /var partition a couple releases ago, so auto configuration is going to be best.
- You only need to set your static management IP for the internal network interface at this time. Leave the other two cards alone for the moment... you will configure them later.

When prompted, reboot your machine:

```
REBOOT
```

Update the your ports using the portsnap tool. This tool is now installed by default in newer BSD versions. Alternatively, you can use dpkg to install the ports you need as well.

```
portsnap fetch && portsnap extract
```

Create your customized kernel file.

Let's start by copying the old kernel file to a new file for editing:

```
cd /usr/src/sys/amd64/conf/ && cp GENERIC ENHANCED
```

Edit your newly created ENHANCED KERNEL and add the options exactly as shown below. You may also want to remove some of the unused devices in the new kernel file as well. Just add a "#" in front of any device you wish to exclude from the new build and it will omit the driver. Be very careful which devices you exclude! Your system may fail to boot if you are not careful!

!!Important!!
The options below need to be in the same area as the rest of the options in your kernel file.

```
edit /usr/src/sys/amd64/conf/ENHANCED
```

```
# Add PF Firewall Support
device pf
# Add PF Logging Support
device pflog
# Add PF firewall synchronization capability for CARP. We will use this later for redundancy.
device pfsync
# Add in the options for IPFW / DUMMYNET (This will be used to control downloads, download queues,etc)
options IPFIREWALL
options IPFIREWALL_VERBOSE
options IPFIREWALL_VERBOSE_LIMIT
options IPFIREWALL_DEFAULT_TO_ACCEPT
options DUMMYNET
# Tune kernel heartbeat. This will help overall performance, but will come at cost of memory & CPU.
options HZ=1000
# Add in ALTQ options for PF. I only use CBQ, but it is nice to have the other options if needed.
options ALTQ
options ALTQ_CBQ
options ALTQ_RED
options ALTQ_RIO
options ALTQ_HFSC
options ALTQ_CDNR
options ALTQ_PRIQ
# Add SMP Support to ALTQ
options ALTQ_NOPCC
```

```
ctrl+[ to save.
```

Now that all of your options have been added, let's build and install the updated kernel. Depending on your processor configuration, this may take a considerable amount of time.

```
cd /usr/src/ && make buildkernel KERNCONF=ENHANCED && make installkernel KERNCONF=ENHANCED
```

Tune kernel, filesystem and network performance

FreeBSD does a great job out of the box tuning network and kernel performance, but we need to further optimize the system for high volumes of traffic. Sysctl allows you to fine tune kernel parameters instantly. Consider sysctl similar to the registry editor, but much more useful. Changes take right after the command is applied.

Add the following parameters into /etc/sysctl.conf to everything sticks at boot:

```
edit /etc/sysctl.conf
```

```
net.inet.tcp.rfc1323=1
kern.ipc.maxsockbuf=16777216
net.inet.tcp.sendspace=1048576
net.inet.ip.intr_queue_maxlen=1000
net.inet.ip.dummynet.hash_size=256
kern.ipc.maxsockbuf=900000000
net.inet.tcp.sendspace=300000000
net.inet.tcp.recvspace=300000000
net.inet.udp.recvspace=300000000
net.inet.tcp.recvspace=1048576
```

ctrl+[to save.

Installing OpenVPN from FreeBSD ports

```
cd /usr/ports/security/openvpn && make install clean
```

Edit and save your new configuration to the openvpn.conf file. Replace x.x.x.x with your IP.

```
edit /usr/local/etc/openvpn/openvpn.conf
```

```
# Specify device
dev tun

# Server and client IP and Pool
server x.x.x.x 255.255.255.0
ifconfig-pool-persist /usr/local/etc/openvpn/ipp.txt
proto udp

# Tell OpenVPN to auth to your Windows AD server
plugin /usr/local/lib/openvpn-auth-pam.so openvpn

# Certificates for VPN Authentication
ca /usr/local/etc/openvpn/keys/ca.crt
cert /usr/local/etc/openvpn/keys/server.crt
key /usr/local/etc/openvpn/keys/server.key
dh /usr/local/etc/openvpn/keys/dh1024.pem
# Have OpenVPN check for revoked certs
crl-verify /home/<yourdir>/keys/crl.pem

# Routes to push to the client
push "route x.x.x.x x.x.x.x"
push "route x.x.x.x x.x.x.x"
# Routes for individual hosts may be required if someone chose to use a common network that exists on the
client and server side.
#push "route x.x.x.x 255.255.255.255"

# Uncomment if you want DNS pushed to clients.  Use with caution!
#push "dhcp-option DNS x.x.x.x"

# Use compression on the VPN link
comp-lzo

# Duplicate CN's allowed
duplicate-cn

# Make the link more resistant to connection failures
```

keepalive 10 86400
persist-tun
persist-key

Run OpenVPN as a daemon and drop privileges to user/group nobody user nobody
group nobody
daemon

reneg-sec 0

ctrl+[to save.

Create directory for CA and adjust permissions:

cp -r /usr/local/share/doc/openvpn/easy-rsa /home/me/
cd /home/me/easy-rsa/
chmod -R 775 /home/me/easy-rsa/

Now build the CA and keys. Answer all of the questions carefully! Please note that you will need to switch shells to ensure these commands work. After you are complete, you may switch back C Shell by typing in "csh".

1) sh
2) . vars
3) ./clean-all
4) ./build-ca
5) ./build-key-server server
6) ./build-key user1
7) ./build-dh
8) cp -r keys /usr/local/etc/openvpn/

Edit syslog to log OpenVPN requests:

edit /etc/syslog.conf:

!openvpn
. /var/log/openvpn.log

ctrl+[to save.

Edit the OpenVPN startup to allow OpenVPN service to start when the system boots: Change the enable from "NO" to "YES" in the OpenVPN startup script under /usr/local/etc/rc.d/:

edit /usr/local/etc/rc.d/openvpn

You now need to set authentication preferences in your OpenVPN configuration:

edit /etc/pam.d/openvpn

auth required pam_radius.so no_warn try_first_pass

ctrl+[to save.

Edit your RC.CONF

Now we are going to ensure all of the necessary services and options start automatically. FreeBSD labels network cards according to manufacturer, so you will need to substitute your interface name in place of the default names I have added (e.g. bge0 may be sk0 on your system). There will also be

additions to this file throughout this document. Ensuring this file does not have any errors is extremely important as the system will NOT boot if it finds a error.

edit /etc/rc.conf

```
# Set your interface IP's
# WAN Interface
ifconfig_bge1="inet <your public IP here> netmask 255.255.255.0"
# Internal Interface
ifconfig bge0=”inet 192.168.1.254 netmask 255.255.255.0”
# DMZ Interface
ifconfig bge2=”inet 192.168.2.254 netmask 255.255.255.0”
# Enable BIND forwarder at boot
named_enable="YES"
# Enable PF at boot
pf_enable="YES"
# Enable pf logging
pflogd_enable="YES"
# Ensure your BSD box is set to gateway mode
gateway_enable="YES"
# SSHv2 access
sshd_enable="YES"
# Enable additional security logging at boot
accounting_enable="YES"
# Fix bug in PF with Active/Passive Proxy requests
ftpproxy_enable="YES"
ftpproxy_flags="-r"
# Enable OpenVPN on boot
openvpn_enable="YES"
# Enable ipfw/dummynet
firewall_enable="YES"
firewall_script="/usr/local/etc/ipfw.rules"
```

ctrl+[to save.

Install BIND:

cd /usr/ports/dns/bind9 && make install clean

Rename existing named.conf and create a new file with content below:

mv /var/named/etc/namedb/named.conf /var/named/etc/namedb/named.old
edit /var/named/etc/namedb/named.conf

```
options {
directory       "/etc/namedb";
pid-file        "/var/run/named/pid";
dump-file       "/var/dump/named_dump.db";
statistics-file "/var/stats/named.stats";
forwarders { x.x.x.x; };
forward only;
};

zone "." in {
type hint;
file "db.cache";
};
```

```
zone "0.0.127.in-addr.arpa" in {
type master;
file "db.127.0.0";
};
```

ctrl+[to save.

Install DHCP Server:

```
cd /usr/ports/net/isc-dhcp30-server && make install clean
```

Edit your dhcpd.conf in /usr/local/etc/ to reflect your configuration. In this example, I used a standard 192.168.1.0/24 network:

```
edit /usr/local/etc/dhcpd.conf
```

```
# dhcpd.conf
# Add in your domain name
option domain-name "home.tld";
# Set your default lease time
default-lease-time 600;
max-lease-time 7200;
# This is the authoritive DHCP server
authoritative;
# Disable dynamic DNS updates.
ddns-update-style none;
# Log file setup.
log-facility local7;
# Declare your subnet, scope and options
subnet 192.168.1.0 netmask 255.255.255.0 {
  range 192.168.1.70 192.168.1.90;
  option domain-name-servers 192.168.1.254;
  option routers 192.168.1.254;
  option broadcast-address 192.168.1.255;
}
```

ctrl+[to save.

Install ftp-proxy for PF.

```
cd /usr/ports/ftp/ftp-proxy && make install clean
```

Configure DUMMYNET and IPFW

Configure DUMMYNET for additional traffic shaping. As I mentioned earlier, this will be used in conjunction with PF ALTQ to ensure bidirectional QoS. There are many that think enabling both DUMMYNET and FS QoS is a bad idea - ignore them… this solution is completely sound. You can use my example below as a starting point:

```
edit /usr/local/etc/ipfw.rules
```

```
IPF="ipfw -q add"
ipfw -q -f flush
ipfw pipe 1 config mask src-ip 0x00000000 bw 10Mbit/s
ipfw add 10 pipe 1 all from any to any xmit bge1
#Add any deny rules in here:
ipfw add 1200 allow all from any to any
# Block spammer net (add whatever net's you want in here)
ipfw add 1300 deny ip from x.x.x.x/xx to any
```

ipfw add 65535 deny all from any to any

ctrl+[to save.

Load the rules:

sh /usr/local/etc/ipfw.rules

Verify DUMMYNET is functioning properly. You should now see queuing for your internal hosts. Everything will now be placed into per host bit buckets:

ipfw pipe show

```
fwhome# ipfw pipe show
00001:  10.000 Mbit/s    0 ms   50 sl. 1 queues (1 buckets) droptail
    mask: 0x00 0x00000000/0x0000 -> 0x00000000/0x0000
BKT Prot ___Source IP/port____ ____Dest. IP/port____ Tot_pkt/bytes Pkt/Byte Drp
  0 tcp    192.168.1.254/22       192.168.1.88/38163 22731 32539104  1 1500   0
fwhome#
```

Bit bucket for host 192.168.1.88

Create your PF configuration file

Now let's configure the main PF component. You will need to adjust the interface names to reflect what you have installed in your server (e.g. sk0, eth0, etc). You can find out what names FreeBSD has associated with your network cards by running: ifconfig | more.

edit /etc/pfstrong.conf

```
### Define Interfaces ####
# External Interface
ext_if = "bge1"
# DMZ Interface
#dmz_if = "bge2"
# Internal Interface - All internal traffic and nets
int_if = "bge0"
# VPN Interface - OpenVPN tunnel for VPN clients
vpn_if="tun0"

### Macros.  These save significant time if you are creating more than a couple rules. ###
# Private nets
privnets = "{ 127.0.0.0/8, 192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8 }"

# Log all external traffic
set loginterface $ext_if

### Optimization options.  These options are useful for fine tuning firewall performance. ###

# Protect firewall from buffer overflow and DDOS attacks
set limit { frags 30000, states 25000 }
# Default firewall optimization level - Set to conservative to ensure proper state handling
set optimization conservative

# Normalize and reassemble traffic on interfaces - Enable this on all interfaces to ensure all packets are normalized.  This will help protect against certain types of frag attacks.
scrub in all

### ALTQ - Traffic Shaping ###
```

```
# Traffic shaping on external interface (Create as many queues as you need here)
altq on $ext_if priq bandwidth 10Mb queue {web, misc}
queue web priority 7
queue misc priority 1 priq(default)

### NAT ###

# Enable NAT
nat on $ext_if from $int_if:network to any -> ($ext_if) round-robin sticky-address

# Fix PF FTP NAT Issue
nat-anchor "ftp-proxy/*"
rdr-anchor "ftp-proxy/*"

rdr on $int_if proto tcp from any to any port 21 -> 127.0.0.1 port 8021

# Redirection rules for internal hosts

# RDR SSL request to DMZ Web Host
rdr on $ext_if proto tcp from any to any port 443 -> 192.168.2.88 port 443

### Tables ###

# Add explicitly denied hosts into the following tables
table <spammers> persist file "/etc/spammers"
table <spamhaus> persist file "/tmp/drop.lasso"

### RULES ###

# Block all connections by default
block log all

# Allow all traffic to loopback interface.  Blocking this is useless.
pass quick on lo0 all

# Allow all traffic on internal interface.  This can be restricted if needed.
pass quick on $int_if all

# Allow VPN traffic to all internal interfaces.  This can be restricted if needed.
pass quick on $vpn_if all

# Anchor for ftp-proxy
anchor "ftp-proxy/*"

# Prevent hijacking on external and internal interfaces
antispoof for $int_if
antispoof for $ext_if

# Block all connections to external interfaces from bogus nets
block drop in on $ext_if from $privnets to any
block drop in on $ext_if from any to $privnets

# Pass HTTPS
pass in on $ext_if inet proto tcp from any to any port 443 modulate state queue web
# Passive mode shim for FTP issues with PF
pass in on $ext_if inet proto tcp from any port 20 to $ext_if port 55000 >< 57000 user proxy modulate state
queue web
# Webmin access ##DO NOT ENABLE!  DEBUGGING ONLY!##
```

```
#pass in on $ext_if inet proto tcp from any to any port 10000 flags S/SA keep state queue web
# NTOP stats for BSD firewall ##DO NOT ENABLE!  DEBUGGING ONLY!##
#pass in on $ext_if inet proto tcp from any to any port 3000 flags S/SA keep state queue web
# Allow OpenVPN traffic from any net - OpenVPN client must have 2 factor auth to complete the connection!
pass in on $ext_if proto udp from any to port 1194 modulate state queue misc

#### Basic rules ####

# Block all ICMP traffic regardless of source
block in on $ext_if inet proto icmp all

# Drop connections from known spammer networks
block in on $ext_if from {<spammers>} to any

# Pass out all traffic on external interface
pass out on $ext_if keep state queue (web, misc)
```

ctrl+[to save.

Now let's see how pf is working:

pftl -s info

```
fwhome# pfctl -s info
Status: Enabled for 2 days 15:55:03          Debug: Urgent

Interface Stats for nfe0              IPv4              IPv6
  Bytes In                       812636978                 0
  Bytes Out                      107375342                 0
  Packets In
    Passed                          670362                 0
    Blocked                          72575                 0
  Packets Out
    Passed                          520325                 0
    Blocked                              0                 0
```

pfctl -s info showing summarized firewall statistics

Here are some other useful PF commands that you will use when managing your firewall:

Enable PF: pfctl –e
Disable PF: pfctl –d
Evaluate effectiveness of rulesets by connection: pfctl –si
Evaluate memory pool usage: vmstat –m
Check state opening duration: pfctl –st
Evaluate PF rule ordering: pftcl –gsr
Evaluate which rules are being evaluated: pfctl –sr
See which PF NAT rules loaded: pfctl –s nat
See which PF RULES loaded: pfctl –s rules
See everything that is loaded: pfctl –s all
Load PF with a specific configuration file: pfctl –f /etc/pfstrong.conf

You will also want to monitor your PF interface on occasion. Here is how to do it:

tcpdump -n -e -ttt -r /var/log/pflog

This command will allow you to flush everything and reload the firewall:

pfctl -Fa -f /etc/pfstrong.conf

And finally, here is an example of monitoring the PF interface with some port and address filtering:

tcpdump -n -e -ttt -r /var/log/pflog port 443 and host x.x.x.x

PF allows you to create lists for blocking nets, users, spammers, etc. In this section we will create a Spamhaus drop list that contains known spammer nets. Download the latest drop.lasso from Spamhaus and paste the contents into the /tmp/lasso.

```
mkdir /tmp/lasso
chmod –R 770 /tmp/lasso
touch /etc/spammers
touch /tmp/drop.lasso
```

Install NTOP & PFTOP

Install NTOP and PFTOP to monitor traffic and performance. You will use these very often to track problem hosts down or just to monitor performance. After you are done doing the make install for NTOP, you will need to change the enable status in its startup file from "NO" to "YES".

```
cd /usr/ports/sysutils/ntop && make install clean
cd /usr/ports/sysutils/pftop && make install clean
```

Run PFTOP to see your firewall connections:

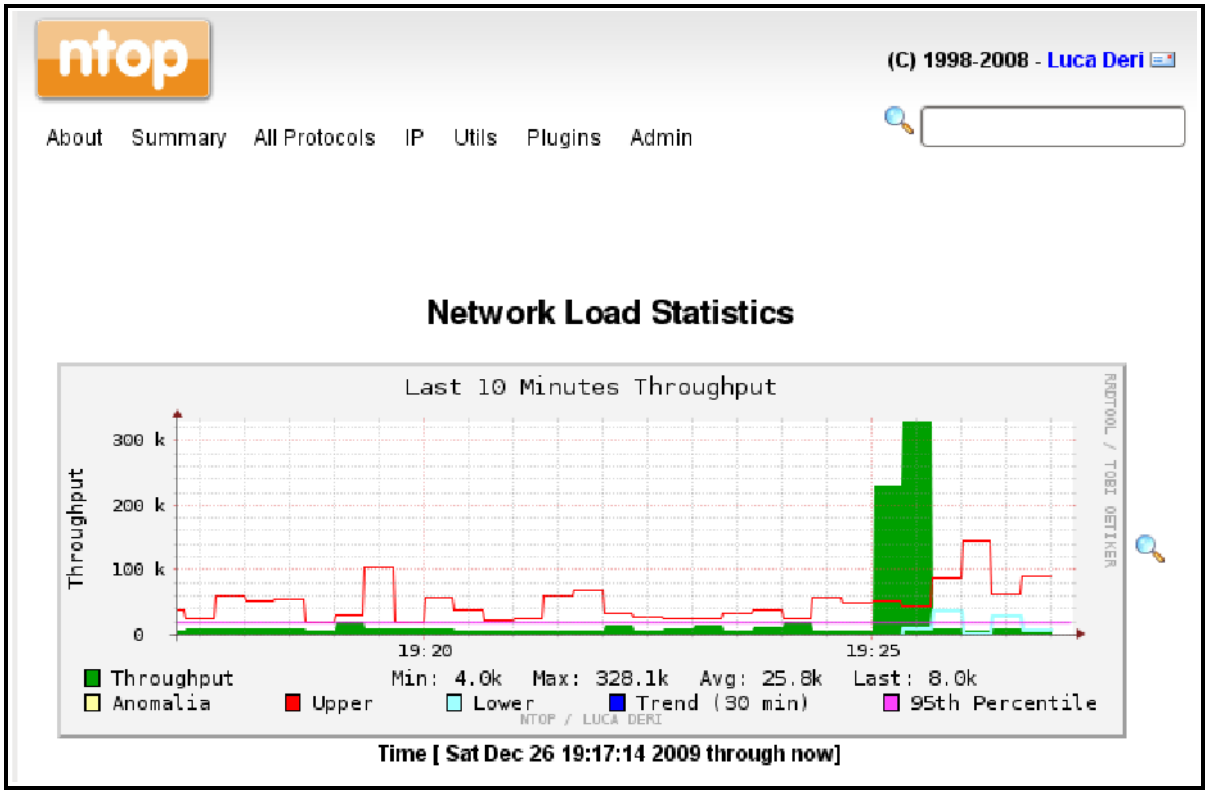
```
pftop
```

```
pfTop: Up State 1-5/5, View: default, Order: none, Cache: 10000

PR      DIR SRC                     DEST                    STATE                  AGE       EXP       PKTS    BYTES
tcp     In  192.168.1.88:46380      [redacted]              ESTABLISHED:ESTABLISHED  01:41:34  04:59:28   368    47883
```

PFTOP displays all active connections in a live format

Now, open NTOP and you should now see traffic statistics. NTOP is a is very extensive tool and allows you to monitor all aspects of your connectivity. It tracks both inbound/outbound connections and ports. The NTOP web interface will be accessible via port 3000... just type in the internal interface IP address of the FreeBSD firewall appliance followed by the port (e.g. 192.168.1.88:3000). After connecting to the web interface, you should see pages similar to the one below:



NTOP capturing network load statistics

Setting up IAS for AD Authentication

Now, we need to Install and configure IAS on your Active Directory Server:

1) Install IAS (Internet Authentication Service) on your Active Directory Server through add remove programs.
2) Open IAS mmc console.
3) Right click on Internet Authentication Service (Local).
4) Right click RADIUS Clients > New Radius Client > Enter your Friendly Name > Enter your Client Address (IP or DNS) > Next.
5) Client-Vendor "Radius Standard" > Enter your shared secret > Finish.
6) Right click on Remote Access Policy > New > Remote Access Policy > Next > Setup a custom policy > Policy name: openvpn > Next > Policy conditions > Add > Select "NAS Identifier" > Add > Type a word or wildcard "OpenVPN" > Grant Remote Access Permission > Next > Edit Profile > Ensure only "Unencrypted Authentication" is selected > Apply.
7) Create user accounts in Active Directory.

!IMPORTANT!
Be sure to enable your VPN users for Remote/Dial-In access in Active Directory!

Patches and Updates to FreeBSD

Let's also make sure our server is up to date and fully patched:

freebsd-update fetch
freebsd-update install
Verify: uname –a

And if you need to rollback the updates – no problem:

freebsd-update rollback

Restart your FreeBSD server and verify services all of the service started properly.

REBOOT

Checking Service Status

After the startup completes, run:

pfctl –s info
ps –aux | grep openvpn
ipfw pipe show

Client OpenVPN Configuration

Next, let's update the OpenVPN client configuration. The user will need their keys in proper directory for the connection to work. Use something like FileZilla to securely transfer the necessary files to your client OpenVPN keys directory. You will need the .crt, .key and .conf file in this location for things to work properly. Now, let's create the client configuration file (openvpn.conf):

touch /etc/openvpn/openvpn.conf

client
Tunnel mode opposed to tap
dev tun
Use UDP instead of TCP/IP

```
proto udp
# Port 1194 is the default.  If you used a different port on the server, change it here
remote xxx.xxxxxx.com 1194
resolv-retry infinite
nobind
persist-key
persist-tun
ca ca.crt
cert ian.crt
key ian.key
# Require password auth along side of the certs
auth-user-pass
reneg-sec 0
ns-cert-type server
comp-lzo
verb 3
route-method exe
# This will fix some issues with Windows clients and adding static routes
route-delay 2
# Adjust keepalives to stop annoying timeouts
keepalive 10 86400
ping-timer-rem
```

Install Apache22 and Nagios

During the initial installation of apache22, the installation process will prompt you for a list of options – just continue by selecting "OK".

cd /usr/ports/www/apache22 && make install clean

After the Apache22 installation is complete, you will need to modify the startup script to run the services on boot. This is done by changing the following options from "NO" to "YES". Note that Apache22_fib remains disabled.

Edit /usr/local/etc/rc.d/apache22

```
[ -z "$apache22_enable" ]       && apache22_enable="YES"
[ -z "$apache22limits_enable" ] && apache22limits_enable="YES"
[ -z "$apache22limits_args" ]   && apache22limits_args="-e -C daemon"
[ -z "$apache22_http_accept_enable" ] && apache22_http_accept_enable="YES"
[ -z "$apache22_fib" ] && apache22_fib="NO"
```

Changing the startup behavior in the Apache22 startup script

ctrl+[to save.

Change httpd.conf to reflect new Nagios configuration:

Edit /usr/local/etc/apache22/httpd.conf

Add this under Directory in /usr/local/etc/apache22/httpd.conf:

```
DocumentRoot "/usr/local/www/apache22/data"

#
# Each directory to which Apache has access can be configured with respect
# to which services and features are allowed and/or disabled in that
# directory (and its subdirectories).
#
# First, we configure the "default" to be a very restrictive set of
# features.
#

<Directory "/usr/local/www/nagios">
    Order deny,allow
    Deny from all
    Allow from 192.168.1.88
</Directory>

<Directory />
    AllowOverride None
    Order deny,allow
    Deny from all
</Directory>
```

Adding in the Nagios directory and restricting to a particular host

Add this under alias_module in /usr/local/etc/apache22/httpd.conf:

```
#
# ScriptAlias: This controls which directories contain server scripts.
# ScriptAliases are essentially the same as Aliases, except that
# documents in the target directory are treated as applications and
# run by the server when requested rather than as documents sent to the
# client.  The same rules about trailing "/" apply to ScriptAlias
# directives as to Alias.
#
ScriptAlias /cgi-bin/ "/usr/local/www/apache22/cgi-bin/"
ScriptAlias /nagios/cgi-bin/ "/usr/local/www/nagios/cgi-bin/"
Alias /nagios/ /usr/local/www/nagios/
```

Adding in the script alias for apache

Add this into the cgi-bin section in /usr/local/etc/apache22/httpd.conf:

```
#
# "/usr/local/www/apache22/cgi-bin" should be changed to whatever your ScriptAliased
# CGI directory exists, if you have that configured.
#

<Directory "/usr/local/www/nagios/cgi-bin">
    Options ExecCGI
    AuthName "Nagios Access"
    AuthType Basic
    AuthUserFile /usr/local/etc/nagios/htpasswd.users
    require valid-user
</Directory>

<Directory "/usr/local/www/apache22/cgi-bin">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
</Directory>
```

Adding in Nagios directory to the CGI configuration in Apache

ctrl+[to save.

Add Apache22 and Nagios (we will add Nagios in the next steps) to rc.conf:

edit /etc/rc.conf

```
# Enable Apache 22
apache22_enable="YES"

#Enable Nagios
nagios_enable="YES"
```

Adding Apache and Nagios into FreeBSD startup

ctrl+[to save.

Install Nagios and select all default installation options.

cd /usr/local/ports/net-mgmt/nagios && make install clean

Edit the Nagios startup script to start service automatically. Change "NO" to "YES":

```
[ -z "${nagios_enable}" ]       && nagios_enable="YES"
[ -z "${nagios_configfile}" ]   && nagios_configfile="/usr/local/etc/nagios/nagios.cfg"
```

Changing the Nagios startup script in /usr/local/etc/rc.d

Now we need to rename all of the Nagios sample config files:

cd /usr/local/etc/nagios

ensure all of the .sample extensions have been removed. Use the mv command as shown below to accomplish this. Do not forget to rename the files in the objects directory as well.

mv nagios.cfg.sample nagios.cfg

Start Apache and Nagios:

sh /usr/local/etc/rc.d/apache22 start

sh /usr/local/etc/rc.d/nagios start

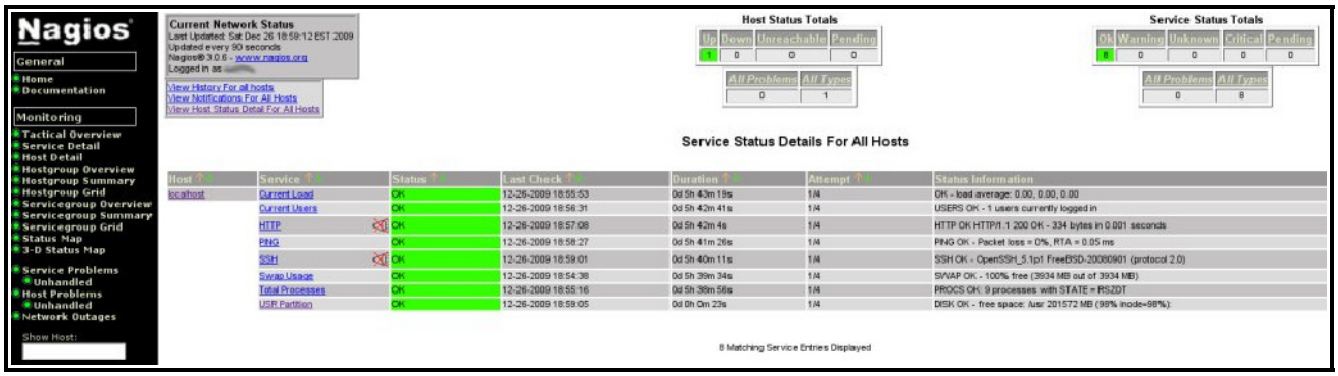
Open the Nagios Dashboard via your web browser (replace "x" with your IP):

http://192.168.1.x/nagios/index.html

You should be greeted with something similar to the picture below:

Nagios Dashboard from Firefox

Let's select "Service Detail" on the left menu bar. You should see your localhost services now:



Nagios Service Detail from Firefox

Installing the WEBMIN System Administration Suite

Install WEBMIN from BSD ports:

```
cd /usr/ports/sysutils/webmin && make install clean
```

Modify startup script. Change "NO" to "YES":

```
webmin_enable=${webmin_enable:-"YES"}

. /etc/rc.subr

name=webmin
rcvar=`set_rcvar`

procname=/usr/local/bin/perl
pidfile=/var/log/webmin/miniserv.pid
required_dirs=/usr/local/etc/webmin
command=/usr/local/etc/webmin/webmin-init start

load_rc_config ${name}
run_rc_command "$1"
```

Modifying the WEBMIN startup script

Add WEBMIN to /etc/rc.conf:

```
# Enable Webmin
webmin_enable="YES"
```

Adding WEBMIN to /etc/rc.conf

Now let's see how WEBMIN works:

sh /usr/local/etc/rc.conf/webmin start

Open the web page to verify correct configuration (my example uses HTTPS):

That is it! You should now have a full featured firewall appliance that is both customizable and reliable. Please be on the lookout for future revisions of this document which will include Squid Proxy, Dans Guardian and Snort.