Network Communications

Contents

ⓘ **NOTE:**
This is a **DRAFT** document which may contain errors!
Help us improve and expand this site.
Please see the **Contrib** section for more details about joining the OpenIndiana Documentation Team.

< Place holder for Introduction content >

1 PPP

< Place holder for content >

2 PPPoE

< Place holder for content >

3 Email

< Place holder for content >

4 WWW server

- Apache
- nginx

5 Firewalls

Firewalls are used to filter network traffic based on rules set by the system administrator. Firewall can protect your personal computer or whole company's network from unauthorized network while allowing passage of legitimate network traffic.

5.1 IP Filter

OpenIndiana comes with built-in firewall, IP Filter. IP Filter is the stateful packet filtering and network address translation (NAT) mechanism. IP filter can filter any kind of traffic based on source or destination IP address or pool of IP addresses, source or destination ports, interface or direction of the network traffic. OpenIndiana IP Filter is derived from open source IPFilter software.

ⓘ **NOTE:**
To manage IP Filter rules one must assume a role that includes IP Filter Management profile rights or superuser (**root**).

5.2 IP Filter configuration

IP Filter is configured by loadable configurations files stored in `/etc/ipf`. One can create several configuration files in `/etc/ipf` for firewall configuration:

- **ipf.conf** - stores packet filtering rules
- **ipnat.conf** - defines NAT rules
- **ippol.conf** - address pool configuration

If IP Filter SMF service is enabled then the configured rules will be automatically loaded at every boot time of the operating system.

5.3 Packet Filtering

IP Filter ruleset can be configured with the ipf(1M) or `/etc/ipf/ipf.conf` file. Rules are processed by the *"the last matching rule wins"* logic. This means that packet passing the IP Filter ruleset from the beggining and the action of the last rule that matched the packet is applied. There are two exceptions, which change this processing. The first one is the use of **quick** keyword, which will apply the rule on the packet and stop further filter rules checks. Another exception is the **group** keyword, which matches packet. Only rules with **group** keyword are used for packet processing.

5.3.1 Filtering rules syntax

The following format is used to create filtering rules:

action [in|out] option keyword, keyword..

Every rule begins with the action. Action can be one of these:

- **block** - denies packets from passing the filter
- **pass** - allows packets to pass the filter
- **log** - logs the packet. ipmon(1M) is used to view the log file.
- **count** - counts packet into the filter statistics. Use ipfstat(1M) to display the statistics.
- **skip** number - skips the filter over number filtering rules
- **auth** - user program is requested to perform packet authentification in order to decide if the packet should be passed or not

Following the action, the next word is **in** or **out**. This determines in which direction rules are applied, e.g incoming or outgoing packets.

The option keyword is next. One can choose from:

- **log** - logs the packet is the packet matched the rule. Use ipmon(1M) to view the log.
- **quick** - rule with quick keyword is executed if packet matches it. No further rules checking is done.
- **on** interface - rule is applied only on interface in both directions
- **dup-to** interface - packet is copied and sent out on interface to specified IP address
- **to** interface - packet is moved to an outbound queue on interface

Next are the keywords that determine if the packet matches the rule. The following keywords shown here can be used:

- **tos** - packet is filtered based on the type-of-service value written as decimal or hexadecimal integer.
- **ttl**- packet is matched based on its time-to-live value.
- **proto** - used to match a specific protocol. Any protocol name from /etc/protocols or its decimal representation can be used.
- **from/to/all/any** - matches either source or destination IP address of the packet and the port number. All accepts packet from any source to any destination.
- **with** - matches specified attributes associated with the packet. Inserted not/no in front of the keyword matches the packet only if the option is not present.
- **flags** - filters based on TCP flags that are set.
- **icmp-type** - filters based on ICMP type.
- **keep** keep-options - determines whetever state should be kept for a packet. state stores information about the session and can be kept on TCP, UDP, and ICMP packets. The frags keeps information about packet fragments and apllies them to later fragments. This option allows matching packets to pass without further ruleset evalution.
- **head** number - creates new group for filtering rules denoted by number.
- **group** number - adds the rule to group number. The default group value is 0.

In the following example we will block all incoming packet on igb0 from 10.0.0.0/8. This rule should be included in one's ruleset:

```
block in quick on igb0 from 10.0.0.0/8 to any
```

5.3.2 Address pools

Address pools group multiple IP addresses/networks into a single reference that can be used in IP Filter rules.

6 NAT (Network Address Translation)

NAT is used in case when one needs to do address or port translation. This happens when one wants to connect multiple computers at home and share the network connection or when one wants to do port forwarding. NAT on OpenIndiana is set up in `/etc/ipf/ipnat.conf` and work regarding NATs is done with ipnat(1M).

6.1 NAT rules syntax

To create NAT rules use the following syntax:

command interface-name parameters

Every rule begins with command from one of these:

- **map** - maps one IP address or network to another IP address or network.
- **rdr** - redirects packet from one IP address and port to another IP address and port.
- **bimap** - creates bidirectorial NAT between an external and an internal IP address.
- **map-block** - establishes static IP address-based translation.

Interface named is used after command, e.g. **igb0**.

To determine NAT configuration one has to use one of the following parameters:

- **ipmask** - designates the network mask.
- **dstipmask** - designates the address ipmask is translated to.
- **mapport** - designates TCP or UDP protocols along with range of ports.

Assuming we have an external IP address 10.0.0.1/24 on interface eg1000 and an internal range of 192.168.1.0/24. The example NAT rule would look like this:

```
map eg1000 192.168.0.0/24 -> 10.0.0.1/24
```

> ⓘ **NOTE:**
> NAT is not usable with IPv6 IP filter as NAT is deprecated in IPv6. NAT can be only used with IPv4 addresses.

7 Advanced Networking

< Place holder for content >

8 Crossbow

< Place holder for content >

9 Hipster as a NAS

OpenIndiana provides several ways to share data with network clients.

Implementation	Description
CIFS	Kernel based SMB file sharing solution offering ZFS integration, ease of use, and relatively simple configuration.
SAMBA	Modern userland based SMB file sharing solution providing support for newer SMB protocols (SMB 2.1) and better compatibility with modern Windows clients.
NFS	The Network File System was originally developed by Sun Microsystems.

ⓘ **NOTE:**
ITEMS TO WRITE ABOUT:
For a variation of configuring a home NAS - this could be done virtually as well

- Running OI as a VMware EXSI guest
  - Local storage hardware is passed through to the OI guest and then shared via ISCSI, CIFS, NFS, etc.

For help writing this section, see the following OpenSolaris references:

- Setting Up an OpenSolaris NAS Box
- Getting Started With the Solaris CIFS Service
- How to enable guest access to a Solaris CIFS share
- Solaris CIFS Service Troubleshooting
- What's New With Solaris CIFS
- CIFS Technical References

Also have a look at the OpenSolaris CIFS Administration Guide

9.1 Hipster as a CIFS server (workgroup)

< Placeholder for introduction content >

9.1.1 Commands used

- `sharemgr` - configure and manage file sharing
- `smbadm` - configure and manage CIFS local groups and users, and manage domain membership
- `zfs` - configures ZFS file systems
- `passwd` - change login password and password attributes
- `chown` - change file ownership

9.1.2 Configuration steps

Start by listing available storage pools.

```
zfs list
```

```
NAME                           USED   AVAIL   REFER   MOUNTPOINT
storage                        498K    899G     19K   /storage
```

Create your ZFS dataset to be shared via CIFS/SMB.

```
zfs create -o casesensitivity=mixed -o sharesmb=on storage/backup
```

Start the CIFS service.

```
svcadm enable -r smb/server
```

Join the CIFS server to a workgroup.

```
smbadm join -w WORKGROUP
```

Configure PAM authentication for the CIFS service.

```
echo "other password required pam_smb_passwd.so.1 nowarn" >> /etc/pam.conf
```

Reset the password for the local user accounts which will be used for remotely accessing the CIFS/SMB share.

```
passwd <user_account>
```

Set the share name to be used for the CIFS/SMB share.

```
zfs set sharesmb=name=backup storage/backup
```

Change the ownership of ZFS dataset to the user account which will be used for remotely accessing the CIFS/SMB share.

```
chown -R <user_account> /storage/backup
```

Verify everything is all set to go.

```
sharemgr show -vp
```

```
default nfs=()
smb smb=()
    * /var/smb/cvol  smb=() ""
              c$=/var/smb/cvol       smb=(abe="false" guestok="false")      "Default
               ↪  Share"
zfs smb=()
zfs/storage/backup smb=()
      backup=/storage/backup
```

You can create additional CIFS datasets using the following 4 commands.

- `zfs create -o casesensitivity=mixed -o sharesmb=on <pool_name/dataset_name>`
- `zfs set sharesmb=name=<new_share_name> <pool_name/dataset_name>`
- `chown -R <user_account> <path_to_dataset>`
- `sharemgr show -vp`

9.2 Hipster as a CIFS server (domain)

< Placeholder for introduction content >

9.3 Hipster as a SAMBA server

< Place holder for content >

9.4 Configuring NFS client connectivity

OpenIndiana has built in NFS client support and automatic mounting capabilities for using remote NFS shares.

9.4.1 Manually mounting remote NFS shares

The `mount -F nfs` command is used to manually mount a remote share to a specified location. Two forms of remote host and path are supported, `host:/remote/path` and `nfs://host/remote/path`. For example, to mount the remote folder `example` on the server `hipster` to the local path `/mnt/example`, this command could be used:

```
# mount -F nfs nfs://hipster/example /mnt/example/`
```

9.4.2 Automatically mounting remote NFS shares

Automounting is provided by the *autofs* service. This enables automatic on demand mounting and unmounting of remote NFS shares. Compared to manually mounting NFS shares, this saves time and doesn't require root access once set up. Mounting specific (direct) remote shares requires configuring the `/etc/auto_master` and `/etc/auto_direct` files.

> ⓘ **NOTE:**
> *autofs* has many other features which are likely to be useful for larger networks with multiple users, such as automatic mounting of user home directories. For more information, see Solaris Network Services documentation.

Before automounting can be used, the autofs daemon must be running. In OI it should be enabled by default:

```
# svcs system/filesystem/autofs
STATE          STIME    FMRI
online         14:59:45 svc:/system/filesystem/autofs:default
```

If it is not already running, it can be enabled with `svcadm enable system/filesystem/autofs`.

The paths for direct NFS shares are configured in `/etc/auto_direct`. For this file to be read by autofs, a new line must first be added to the end of `/etc/auto_master`:

```
# Master map for automounter
#
+auto_master
/net            -hosts          -nosuid,nobrowse
/home           auto_home       -nobrowse
/-              auto_direct
```

The `/etc/auto_direct` file can then be created and populated. Entries consist of the local path to mount the NFS share, optional mount options, and the remote NFS path. Both the forms in the previous section for specifying the remote path are supported. For example, this configures two automatically mounted shares, one using the `ro` option (for read-only):

```
/mnt/example nfs://hipster/example
/mnt/reference -ro hipster:/reference
```

For changes to these files to take effect, reboot or run the `automount` commmand.

The `mount` command is used without arguments to list all current mounts, including any automatically mounted NFS locations and mount time (when the automatic mount was triggered).

9.5 Configuring CIFS/SMB client connectivity

< Place holder for content >

9.5.1 Adding a remote share using the Linux smbclient

- Accessing an SMB Share With Linux Machines

9.5.2 Adding a remote share using the KDE Dolphin file manager GUI

- In the left hand pane click *Network*
- In the right hand pane click *Add Network Folder*
- The Network Folder Wizard opens
- Select the radio button for *Microsoft Windows network drive* and click next
- Specify a name for the share - can be anything - this is just a label
- Specify the remote CIFS/SMB server name (or IP address)
- Specify the share name of the remote CIFS/SMB share
- Click the save and connect button
- You'll be prompted for a remote username and password
- Ensure the checkbox is marked to save credentials or you'll be asked for everything you do.

9.5.3 Adding a remote share using a Windows client

< Place holder for content >

9.6 Hipster as an NFS server

< Placeholder for introduction content >

9.6.1 Manual pages

You can read more about the commands and files used or mentioned with the `man` pages, and at the provided links in the document.

- dfstab(5)
- share_nfs(8)
- share(8)
- shareall(8)
- svcadm(8)
- unshare(8)

For example to view the full documentation for `share(8)` you can use this command:

```
$ man share.8
```

9.6.2 Sharing a folder with NFS

To create a read-only temporary NFS share on an existing folder in this example `/nfs/example` you can use the following share(8) command:

```
# share -d example-share -F nfs -o ro /nfs/example
```

The flags provided can be described in a table such as the following:

Flag	Description
-d	Adds a description for the share in this case `example-share`
-F	Filesystem type, in this case NFS.

Flag	Description
-o	Options for the NFS share in this case `ro`

If you want the share to be persistent, you can write it to the dfstab(5) by using the `-p` flag on the share(8) command.

You can then use the share(8) command with no arguments to list shares as follows:

```
$ share
-               /nfs/example   ro   "example-share"
```

After this you can enable the `network/nfs/server` service with svcadm(8) using the following command:

```
# svcadm enable network/nfs/server
```

And then use shareall(8) to share it on the network as follows:

```
# shareall
```

9.6.3 Using an access list

An access list argument is a colon separated list, the access list may include the following:

- A hostname, which must be represented as a fully qualified DNS or LDAP name.
- A netgroup
- A domain name suffix
- A network, IP address or IP subnet

This is an example of creating an NFS share that's read-only for everyone, but provide read-write and root access to a single IP address in this case `192.168.0.100`, IP addresses are prefixed with `@` as they are in the network components:

```
# share -F nfs -o ro,rw=@192.168.0.100,root=@192.168.0.100 /nfs/example
```

9.6.4 Other useful NFS options

Option	Description
nosuid	Ignore attempts of clients setting suid bits.
none=*access_list*	Deny all access to clients that match the access list.

9.6.5 Removing an NFS share

You can use the unshare(8) utility with the NFS share folder as shown here:

```
# unshare /nfs/example
```

9.6.6 Securing NFS with Kerberos

< Place holder for content >

10 Hipster as an NIS server

< Place holder for content >

11 Hipster as an LDAP server

< Place holder for content >

12 Hipster as a DHCP server

< Place holder for content >

13 Hipster as an FTP server

< Place holder for content >

14 Hipster as a DNS server

see https://docs.oracle.com/cd/E23824_01/html/821-1455/dnsref-31.html specifically pkg install pkg:/service/network/dns/bind

15 Hipster as a NTP server

< Place holder for content >

16 Hipster as a INETD server

< Place holder for content >