FreeBSD iSCSI Primer

BY JASON TUBNOR

We all hear about Network Attached Storage (NAS) being able to provide additional storage for devices on your network. However, the protocols for this storage may not be appropriate for all use cases.

Welcome to the world of Storage Area Network (SAN). Typically, these are found more in enterprises than the home or small business, but that does not mean that they shouldn't be used in such a situation. In fact, you probably have a very good use case if you do lots of virtualization with central storage connected to multiple compute devices or have a need to provide block storage to Windows Workstations used for engineering or graphics design that require more storage than can physically fit within desktop PCs.

Typically, the SAN providers that we hear of in the enterprise space are provided by the likes of Dell EMC, IBM, Hitachi and NetApp to name a few. However, we are spoiled in the FreeBSD space to have a high-performance, iSCSI SAN solution baked right into the base system. Using this, in combination with the powerful ZFS volume manager and file system, we have a flexible, resilient, and fast storage solution that we can make available to network clients. The iSCSI subsystem was implemented and part of the 10.0-RELEASE with numerous performance improvements coming out with the 10.1-RELEASE.

Typically, Storage Area Networks are found more in enterprises than the home or small business, but that does not mean that they shouldn't be used in such a situation.

Internet Small Computer Systems Interface or iSCSI for short is an IP-based protocol for carrying SCSI commands over a TCP/IP ethernet network. It allows the presentation of block device storage to machines distributed across the network. It is flexible enough to even be routed over the internet if required, but the security implications and precautions that need to be considered are out of scope for this text.

In an ideal world, iSCSI should exist within its own Layer 2 physical network where the only communication from compute hosts via dedicated storage interfaces is to that of the storage target. No other general network traffic should exist on this network segment to avoid contention to the storage for this and other compute nodes. In a smaller environment, using a segmented switch with VLANs is an acceptable alternative, however, understand that general network traffic and storage traffic will be competing for interface bandwidth.

The high-level iSCSI terminology is quite simple. There is the initiator (client) and the target (host). The initiator is the active end of the connection whereas the target is passive — it will never try to connect to an initiator.

In the following exercise, we are going to prepare a simple iSCSI configuration on a FreeBSD host to use as a target and provide ZFS **`zvol`** block devices to a FreeBSD and MS Windows initiator.

Our hosts on the network:

```
2001:db8:1::a/64 - FreeBSD ZFS Storage Host (Target)
2001:db8:1::1/64 - FreeBSD Client (Initiator)
2001:db8:1::2/64 - Windows Server 2022 (Initiator)
```

First, we will configure ZFS volumes to present as iSCSI targets for each of the initiators on the storage host. This could also be simply files on a ZFS dataset or UFS partition, but ZFS volumes give you far more control over aspects of the storage, especially ongoing management as data requirements change or in relation to snapshot and replication requirements.

```
zfs create -o volmode=dev -V 50G tank/fblock0
zfs create -o volmode=dev -V 50G tank/wblock0
```

Adjust the **`volblocksize`** attribute when creating the volumes to meet the requirements of the workload that they will be used for. As of 14.1-RELEASE, they will be set to 16K which is a reasonable balance for most workloads.

Create a file **`/etc/ctl.conf`** that is only read/writeable by root. This file will contain secrets, so it is essential that no other group or user has the ability to view or write to this file. Below are the contents that we will add to provide storage points for our initiators:

```
auth-group ag0 {
    chap-mutual "inituser1" "secretpassw0rd" "targetuser1" "topspassw0rd"
    initiator-portal [2001:db8:1::1]
}

auth-group ag1 {
    chap-mutual "inituser2" "hiddenpassw0rd" "targetuser2" "freepassw0rd"
    initiator-portal [2001:db8:1::2]
}

portal-group pg0        {
    discovery-auth-group no-authentication
    listen [2001:db8:1::a]
}

target iqn.2012-06.org.example.iscsi:target1 {
    alias "Target for FreeBSD"
    auth-group ag0
    portal-group pg0
    lun 0 {
        path /dev/zvol/tank/fblock0
```

```
        #blocksize 4096
        option naa 0x4ee0ebaf06a1acee
        option pblocksize 4096
        option ublocksize 4096
    }
}

target iqn.2012-06.org.example.iscsi:target2 {
    alias "Target for Windows"
    auth-group ag1
    portal-group pg0
    lun 1 {
        path /dev/zvol/tank/wblock0
        blocksize 4096
        option naa 0x4ee0ebaf06a1acbb
    }
}
```

Let's break down the file to get an understanding of the what and why of each component:

```
auth-group ag0 {
    chap-mutual "inituser1" "secretpassw0rd" "targetuser1" "topspassw0rd"
    initiator-portal [2001:db8:1::1]
}
```

This is the authorization group that can be used across multiple targets. This example binds the **`auth-group`** to a unique initiator with the address 2001:db8:1::1 and requires it to have mutual authentication. You can simply use CHAP authentication as a 'one way' authentication, but it is recommended that, if supported, you use mutual authentication to ensure that both the initiator and the target are correctly authenticating against each other.

This authentication may be sufficient where the initiator and target reside on the same physical network, but it should not be relied upon as a security control. This type of authentication should be seen as a method of ensuring that only the correct storage is allocated to the initiator. Loosely configured iSCSI targets could make the wrong storage available to a target which could have the undesired effect of overwriting data, partition table, or other meta data, so restricting access to a specific target dataset is essential.

```
portal-group pg0        {
    discovery-auth-group no-authentication
    listen [2001:db8:1::a]
}
```

Portal groups set the target environment offered to the initiators. Here we define a portal group to allow initiators to connect to the target via 2001:db8:1::a and so they can discover the target datasets that include this portal group without having to authenticate first. Typically, in a controlled environment, this would be fine to ensure initiators can find what they need to connect to, but would be undesirable in a more hostile or untrusted environment.

```
target iqn.2012-06.org.example.iscsi:target1 {
    alias "Target for FreeBSD"
    auth-group ag0
    portal-group pg0
    lun 0 {
        path /dev/zvol/tank/fblock0
        #blocksize 4096
        option naa 0x4ee0ebaf06a1acee
        option pblocksize 4096
        option ublocksize 4096
    }
}
```

The meat of the configuration is the target. This will include the **auth-group** and **portal-group** to build up the previously described components for the presentation to the initiator. These can be over-ridden on a per target basis and can be defined without the applicable **group** definition.

The iSCSI Qualified Name (IQN) format takes the form **iqn.yyyy-mm.namingauthority:uniquename**. This is needed for each target definition.

The alias definition is simply a human readable description for the target.

Each target can have multiple LUNs but here we simply have only one LUN per target.

The LUN context allows you to define the characteristics of the LUN. This is important where ZFS is being used on the presented storage within the initiator. While your ZFS volume on the target has a block size of 16KB, the ZFS pool — when created on the initiator — will complain that the block size of the storage is not 4KB or less. It won't stop you from using it, but a message when you execute **zpool status** on the initiator will continually remind you of this. Adjusting the block size attribute to 4096 is not sufficient to remediate the issue. The **pblocksize** and **ublocksize** options will need to specifically be added for a ZFS use case.

Each target can have multiple LUNs but here we simply have only one LUN per target.

Option naa should be explicitly defined for LUNs. This is either a 64- or 128-bit unique hexadecimal identifier. This is important when you are backing VMWare compute onto an iSCSI target to ensure there is no confusions with LUN assignments.

This is now enough to get you up and running and present storage from your target. Enable **ctld** and bring up the daemon:

```
service ctld enable
service ctld start
```

To verify that the storage is presented, you can check that the daemon is listening:

```
# netstat -na | grep 3260
tcp6       0      0 2001:db8:1::a.3260     *.*                    LISTEN
```

Then verify the LUNs are presented using the CAM Target Layer control utility:

```
# ctladm lunlist
(7:1:0/0):<FREEBSD CTLDISK 0001> Fixed Direct Access SPC-5 SCSI device
(7:1:1/1):<FREEBSD CTLDISK 0001> Fixed Direct Access SPC-5 SCSI device
# ctladm devlist
LUN Backend      Size (Blocks)   BS Serial Number     Device ID
  0 block           104857600   512 MYSERIAL0000   MYDEVID0000
  1 block            13107200  4096 MYSERIAL0001   MYDEVID0001
```

Now to configure your FreeBSD initiator. You need to create the **/etc/iscsi.conf** configuration file. This also needs to be set as read/write explicitly for root as it contains secrets:

```
fblock0         {
targetaddress   = [2001:db8:1::a];
targetname      = iqn.2012-06.org.example.iscsi:target1;
initiatorname   = iqn.2012-06.org.example.freebsd:nobody;
authmethod      = CHAP;
chapiname       = "inituser1";
chapsecret      = "secretpassw0rd";
tgtChapName     = "targetuser1";
tgtChapSecret   = "topspassw0rd";
}
```

Each of these attributes are:

- **fblock0** — This is a human readable identifier; it is not related to anything except grouping each of the following configuration items.
- **targetaddress** — The network address of the storage target. This can also be a fully qualified domain name.
- **targetname** — This will align with the corresponding target name that was defined in the **ctl.conf** file
- **initiatorname** — Defining the IQN of the initiator.
- **authmethod** — Can simply be defined on FreeBSD as CHAP. Mutal settings will be assumed if **ChapName** and **ChapSecret** are prefixed with '**tgt**'.
- **chapiname/chapsecret** — Authentication as defined previously in the **ctl.conf** file.
- **tgtChap[Name,Secret]** — Authentication that the target needs to complete the authentication handshake with the initiator.

To enable use, simply issue:

```
service iscsid enable
service iscsictl enable
service iscsid start
service iscsictl start
```

This should then attach the target's storage presentation to the initiator:

```
# iscsictl -L
Target name                                Target portal    State
iqn.2012-06.org.example.iscsi:target1 [2001:db8:1::a]  Connected: da0
```

While we are here, let's look at the simple switches that will be used most frequently with the `iscsictl` command:

- `L` This lists targets mounted to the initiator and where they are connected.
- `Aa` Attach all targets defined in the iscsi.conf file
- `Ra` Remove all targets that are connected to the initiator

```
# gpart create -s GPT da0
da0 created
# gpart add -t freebsd-zfs -a 1M da0
da0p1 added
# zpool create tank da0p1
# zpool list tank
NAME   SIZE  ALLOC   FREE  CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP    HEALTH  ALTROOT
tank  49.5G   360K  49.5G        -         -     0%     0%  1.00x    ONLINE        -
# zpool status tank
  pool: tank
 state: ONLINE
config:

        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          da0p1     ONLINE       0     0     0

errors: No known data errors
```

The manual pages for the configuration, daemons, and control tools are exceptionally well written and can be referenced to get a better understanding of what else is available.

This only touches the surface of the full power that is available with the iSCSI implementation within FreeBSD, but it gives you an idea and practical examples of how it can be harnessed to provide flexible, remote storage options for your computing infrastructure.

JASON TUBNOR has over 28 years of IT industry experience in a vast range of disciplines and is currently the ICT Senior Security Lead at Latrobe Community Health Service (Victoria, Australia). Discovering Linux and Open Source in the mid 1990s, then being introduced to OpenBSD in 2000, Jason has used these tools to solve various problems in organizations that cover different industries. Jason is also a co-host on the BSDNow Podcast.