Protecting Data with ZFS Native Encryption

BY ROLLER ANGEL

ZFS has native support for encrypting datasets which allows you to easily protect data with industry-standard cipher suites. The major benefit to encrypting a dataset on a disk vs full-disk encryption of the disk is that a dataset can be unmounted when not in use, while full-disk encryption requires the disk to be powered down to get the data encrypted while it is at rest. Keep in mind that ZFS native encryption has the concept of loading and unloading keys. Simply unmounting the encrypted dataset is not enough. You must also unload the key associated with that particular dataset. If the key is still loaded, the dataset can be mounted and the data will be available. Unloading the key will make the mount operation fail. Loading the key is a prerequisite to mounting the dataset. Nested child datasets inherit the encryption key of their parent — but they don't have to. Different encryption keys and cipher suites may be used even if the parent dataset uses different encryption settings. Finally, changing keys is as easy as issuing the **`zfs change-key`** command on the dataset.

Those are the basic concepts to get started.

Turning on the encryption parameter for a newly created dataset and setting a key format is enough to get started. If an encryption cipher suite isn't specified, the default of **`aes-256-gcm`** is used. The default is subject to change as new cipher suites get added in the future. The encryption property of an existing dataset is read-only, modifying the property of an unencrypted dataset to turn on encryption isn't allowed. To specify the encryption properties, you need to know what options are available. I suggest reading up on the available options in the **`zfsprops`** man page. Do this by typing the command **`man zfsprops`**. I also recommend reading the man page **`zfs-load-key`**. For our first encrypted dataset, we will start with the default cipher suite, the passphrase key format, and create a dataset called **`secrets`**. I'm using a FreeBSD jail machine I created in my lab for this article called alice. All the jails in my lab exist on the zpool called lab. I've made a zpool available to the jail with the name zroot. From inside the jail I must use the entire path **`lab/alice/zroot`** as the name of my zpool in order to create a dataset within. For contrast, on my laptop I can simply use the name of my zpool and directly create a dataset there. Listed below are the commands to create an encrypted dataset on both the alice jail as well as on my laptop. Like any ZFS dataset, setting a mount point is a

Loading the key is a prerequisite to mounting the dataset.

good idea, just keep in mind ZFS is a layering filesystem so don't use an existing path as the new dataset mount point.

alice jail:

```
zfs create -o encryption=on -o keyformat=passphrase -o mountpoint=/secrets
lab/alice/zroot/secrets
```

my laptop:

```
zfs create -o encryption=on -o keyformat=passphrase -o mountpoint=/secrets zroot/secrets
```

After running the **`zfs create`** command, I'm prompted to enter in a sufficiently long passphrase. Now, I have a mounted dataset with encryption enabled where I can store data that I need to protect. While the dataset is mounted, I can use it like any other unencrypted dataset. When I'm done adding the secret data, I can unmount the **`secrets`** dataset and unload the key together in one command by typing **`zfs unmount -u lab/alice/zroot/secrets`**. To decrypt and mount the data again, I run the command **`zfs mount -l lab/alice/zroot/secrets`**. This will ask me for my passphrase, load the key, then mount the dataset. Omitting the **`-u`** flag in the unmount command will only unmount the dataset, leaving the key loaded. The dataset can still be mounted without prompting for the passphrase with **`zfs mount lab/alice/zroot/secrets`**. To unload the key after the dataset has been unmounted, I run **`zfs unload-key lab/alice/zroot/secrets`**. Now the previous mount command will fail because the key isn't loaded and I didn't provide the **`-l`** flag to ask ZFS to load the key before mounting. To load a key and allow the dataset to be mounted, I run **`zfs load-key lab/alice/zroot/secrets`**. I'm prompted for my passphrase and the previous mount command will now succeed because the key is loaded. To check whether a key is loaded or not, view the properties of the dataset. Some useful properties to look at are displayed when I run **`zfs list -o name,mountpoint,encryption,keylocation,keyformat,keystatus,encryptionroot lab/alice/zroot/secrets`**. The **`KEYSTATUS`** column shows **`available`** meaning the key is loaded. To see all of the dataset properties I can use **`zfs get all lab/alice/zroot/secrets`**.

While the dataset is mounted, I can use it like any other unencrypted dataset.

Next I create a dataset nested below the **`secrets`** dataset with a different cipher suite and key format. This time, I'll use a key file instead of a passphrase. To create a dataset that uses a key file, I first need to generate the key and store it in a file. I do this by typing the command **`dd if=/dev/urandom bs=32 count=1 of=/media/more-secrets.key`**. I used the **`bs=32`** because the key file is required to be 32 bytes long. Also, the output is going to the **`/media`** path because I have mounted a portable USB drive there and have used **`dd`** to generate the key file and store it directly onto the drive. This is so there is no trace of the key file on my machine when I unload the key and I unmount and remove the USB drive. I recommend storing this key file on more than one USB drive as a safeguard in case a USB drive is damaged. Now that the key file has been generated I can create the nested dataset with the AES-256-CCM cipher suite by running **`zfs create -o encryption=aes-256-ccm -o keyformat=raw -o keylocation=file:///media/more-secrets.key -o mountpoint=/secrets/more-secrets lab/alice/zroot/secrets/more-secrets`**. Viewing the proper-

ties of this new dataset I can see the `ENCROOT` column is set to `lab/alice/zroot/secrets/more-secrets`. I can use the same methods to unmount and unload the key as I used on the `secrets` dataset. As I get more datasets and keys, I may want to consider unloading all the keys with `zfs unload-key -a` or I can unload a subset of keys by unloading just the keys for the `secrets` dataset and all descendant datasets with `zfs unload-key -r lab/alice/zroot/secrets`. Conversely, If I want to load a subset of keys for the secrets dataset and all descendant datasets, I run `zfs load-key -r lab/alice/zroot/secrets`.

If I decide later on that the data in this `more-secrets` dataset doesn't need to have a separate key file, and instead, I want it to inherit the settings from its parent dataset `secrets` (switching from custom generated key file to the passphrase configured earlier) all I need to do is run `zfs change-key -i lab/alice/zroot/secrets/more-secrets`. View the properties again and notice that `ENCROOT`, `KEYLOCATION`, and `KEYFORMAT` have all changed. The encryption suite, however, doesn't change because the encryption suite can only be set on the creation of a dataset. Since `more-secrets` is contained within `secrets` it will unmount as part unmouting the `secrets` dataset. Although mounting the `secrets` dataset will not also mount `more-secrets`. That will need to be mounted separately, but the key will only need to be loaded once since they both share the same key. To switch back to using the key file, I run `zfs change-key -o keyformat=raw -o keylocation=file:///media/more-secrets.key lab/alice/zroot/secrets/more-secrets`. If I want to permanently destroy the data in `more-secrets`, I simply unmount the dataset, unload the key, and destroy the key file and any backup copies of the key file I have made. Now, the data is not able to be recovered. I can then run `zfs destroy lab/alice/zroot/secrets/more-secrets` to remove the dataset.

ZFS native encryption allows data to be easily protected with encryption.

One final note I'd like to share is regarding backups of encrypted data. As you have seen, ZFS native encryption allows data to be easily protected with encryption. Snapshots of encrypted datasets can be received on an untrusted backup server in their encrypted form. Without the key, the remote backup server won't be able to mount the dataset. Use the `--raw` flag of the `zfs send` command to help accomplish this. For more details, I suggest reading the man page `zfs-send` to get an idea of how it works and then get a copy of the book *ZFS Mastery: Advanced ZFS* to really dive deep into the specifics and to learn a myriad of techniques to further hone your ZFS skills.

I hope you enjoyed this how-to article and that you begin protecting your sensitive data using the native encryption offered by the amazing ZFS filesystem.

ROLLER ANGEL spends most of his time helping people learn how to accomplish their goals using technology. He's an avid FreeBSD Systems Administrator and Pythonista who enjoys learning amazing things that can be done with Open Source technology — especially FreeBSD and Python — to solve issues. He's a firm believer that people can learn anything they wish to set their minds to. Roller is always seeking creative solutions to problems and enjoys a good challenge. He's driven and motivated to learn, explore new ideas, and to keep his skills sharp. He enjoys participating in the research community and sharing his ideas.