zNcrypt 3.3 Quick Start Guide

Last Updated: November 26, 2013; Copyright©2011-2014 by Cloudera

Overview

This document contains instructions to download and configure zNcrypt. For a full list of zNcrypt 3.3 features, prerequisites, supported Operating Systems, and installation commands, please see the full zNcrypt 3.3 User Guide.

Automating a zNcrypt Deployment

Between testing, installation, and just for plain old fun, Cloudera employees install zNcrypt quite a bit. Because of how much we install zNcrypt, we were even nice enough to create scripts that gloss over most of the nasty details. If you are interested in accessing our development tools or scripts, please use the zNcrypt Express Guide and visit our GitHub Repository, which houses the most up-to-date versions of these wildly helpful tools.

Important Information

Installing the zNcrypt Client will require planned downtime for all processes that are using encrypted data. In addition, the commands contained in this document should only be used by a knowledgeable system administrator. Improper use of these commands could cause irreparable system damage, so it is also recommended that backups be taken prior to proceeding.

If you have any questions or concerns, please don't hesitate to contact support@gazzang.com prior to installing.

Before Installation

Base System Requirements

- Linux kernel 2.6.19 or later (Red Hat™ and CentOS™ can use 2.6.18-92 or later)
- Supported Linux Systems:
  - Red Hat™ 5.4 and above, 6.x
  - CentOS™ 5.4 and above, 6.x
  - Ubuntu™ 11.10 (32-bit only), 12.04, 12.10, 13.04
  - And many more not covered in the guide, please review the full zNcrypt User Documentation for the full list

Administrative Access

In order to enforce the utmost level of security, all zNcrypt commands require administrative (root) access (including installation and configuration). If you do not have administrative privileges on your server, please contact your system administrator for help before proceeding.

Installing Kernel Development Packages

Before installing zNcrypt, we recommend you install the kernel development packages. Each kernel module is compiled specifically for the underlying kernel version. In order for zNcrypt to run as a kernel module, you must download and install the kernel development headers. It is this ability to run as a kernel module which allows zNcrypt to boast high performance metrics while still being completely transparent to user-space applications.

To determine your current running kernel version, run:

```
$ uname -r
```

To install the development headers for your current kernel version, run:

Ubuntu Systems

```
$ sudo apt-get install linux-headers-$(uname -r)
```

Red Hat Enterprise Linux Systems

```
$ sudo yum install kernel-headers-$(uname -r) kernel-devel-$(uname -r)
```

zNcrypt Installation

Adding the Cloudera Public Repository to System

The first step in the installation is adding the Cloudera repository to the package management system.

Ubuntu Systems

Set the system environment variables:

```
$ . /etc/lsb-release
```

Add the Cloudera Repository URL to the apt sources.list file:

```
$ echo "deb https://archive.gazzang.com/ubuntu/stable $DISTRIB_CODENAME main" | sudo tee -a /etc/apt/sources.list
```

Red Hat Enterprise Linux Systems

Create and open the Gazzang repo file with the following command:

```
$ sudo vi /etc/yum.repos.d/gazzang.repo
```

And paste the following text:

```
[gazzang]
name=RHEL $releasever - gazzang.com - base
baseurl=https://archive.gazzang.com/redhat/stable/$releasever
```


```
enabled=1
gpgcheck=1
gpgkey=https://archive.gazzang.com/gpg_gazzang.asc
```

Adding the Cloudera Public Key to Verify Received Packages

Cloudera signs all of our repository packages with a GPG key so that you can ensure that your downloaded packages are authentic and secure.

Ubuntu Systems

Import the Cloudera public key:

```
$ wget -O - https://archive.gazzang.com/gpg_gazzang.asc | sudo apt-key add -
```

Red Hat Enterprise Linux Systems

Import the public key into RPM:

```
$ sudo rpm --import https://archive.gazzang.com/gpg_gazzang.asc
```

Installing zNcrypt

Now that we've prepared the system and installed all of our necessary dependencies, it's time to install the encryption client.

Ubuntu Systems

Install the zNcrypt client through the aptitude package manager:

```
$ sudo apt-get update
```

```
$ sudo apt-get install zncrypt haveged
```

Red Hat Enterprise Linux Systems

Install the zNcrypt client through the yum package manager:

```
$ sudo yum install zncrypt haveged
```

And to ensure that zNcrypt is started next time you reboot your machine, we'll need to add it to the start order:

```
$ chkconfig --level 235 zncrypt-mount on
```

```
$ chkconfig --level 235 haveged on
```

```
$ service haveged start
```


Potential Pitfalls

The most common error that occurs while installing the zNcrypt client is a build failure for the zNcryptFS Kernel Module (packaged inside of zNcrypt). In order to prevent this, please make sure that the kernel development packages are installed for your currently running kernel version.

Once the kernel headers have been installed, you can manually compile the zNcryptFS module for your current kernel version with:

```
$ sudo zncrypt-module-setup
```

If the build process continues to fail, please don't hesitate to contact us at support@gazzang.com.

Registering with the zTrustee Keystore

The first step in using your newly installed zNcrypt client is registration. Registration will alert the Cloudera Key Server that your are planning to store keys in the repository. In addition to readying the key storage, registration is also when you specify your master password(s).

```
$ sudo zncrypt register -o <organization name> --auth <org secret>
```

As you can see above, registration requires the knowledge of two fields:

> -o <organization name>
>
> This is the organization name that you will be registering your client against. Typically, this is just the name of the company that you are representing.
>
> --auth <org secret>
>
> This is the authorization secret tied to the organization above.

***Note**, if you aren't sure about either of the fields above, please contact support@gazzang.com to receive credentials.*

By default, a register command will attempt to contact the default Cloudera SaaS zTrustee Server. If you have an on premise zTrustee server you would like to register against, you can make use of the '-s' flag:

```
$ sudo zncrypt register -o <organization name> --auth <org secret> -s <key mgr hostname>
```

STOP! It is ***extremely*** important that you keep your master password secret and safe. In the event that you lose your master password, you will never be able to recover it, leaving your encrypted data irrevocably locked away.

Encrypting

The instructions below are for a basic configuration of file-level encryption. zNcrypt offers multiple types of encryption with varying levels of configuration. To learn more, contact support@gazzang.com.

About Encryption

In order to encrypt your data, zNcrypt needs to create a separate encrypted file system that will sit on top of your existing Linux file system. During the encryption process zNcrypt will move your existing data into this encrypted file system and create a symbolic link between the original location and the new encrypted data location. Processes will access the data through its original location, which will now be directed (via the symlink) to the encrypted data in its new location, thus creating transparent data encryption.

Preparing for Encryption

In order to prepare for encryption, we will have to set a location to store the encrypted data. In the following example we will use the directory /var/lib/zncrypt/encrypted and /var/lib/zncrypt/mount. If you have certain space/partition requirements please use your best judgement in selecting a different directory, though it is highly recommended that you place the encrypted directory on the same partition as the data you are planning to encrypt.

The syntax for the prepare command is as follows:

```
$ sudo zncrypt-prepare <location to store data> <mount-point for partition>
```

To create our encrypted partition, we will need to create the storage/mount directory:

```
$ sudo mkdir -p /var/lib/zncrypt/encrypted /var/lib/zncrypt/mount
```

Then create the encrypted partition:

```
$ sudo zncrypt-prepare /var/lib/zncrypt/encrypted /var/lib/zncrypt/mount
```

***Note**, the example above uses different locations to store and access the directories. This is to demonstrate the difference between the two directories. It is also possible to store and access the data from the same directory.*

To get a better understanding of what the command above accomplished, you can run:

```
$ df -hT
```

This command will display all of the partition information about your system. You should see an 'eCryptFS' partition now being located on the /var/lib/zncrypt/encrypted directory, and mounted on the /var/lib/zncrypt/mount directory.

To get an in-depth look at the details behind the zncrypt-prepare command (or to use a unique configuration), you can also use the interactive prompt by simply executing:

```
$ sudo zncrypt-prepare
```


Which will launch an interactive console that will discuss each parameter in detail.

Encrypting the Data

Once the encrypted file system is created and initialized, it is ready to hold data.

Warning, prior to running any encryption, it is important that all processes (MySQL, MongoDB, PostgreSQL, etc) that have access to the target data are stopped. Failure to do so could lead to potential data corruption.

All encryption and decryption functionality come from a single command, zncrypt-move. The syntax of a generic encryption command will look similar to the following:

```
$ sudo zncrypt-move encrypt @<category> <directory to encrypt> <encrypted mount-point>
```

With the following parameters:

zncrypt-move

> This is the main command interface for all actions that require moving data either to or from the encrypted file system. To see more about everything you can do with zncrypt-move, please see the zncrypt-move manual ($ man zncrypt-move).

encrypt

> This parameter lets zNcrypt know which direction to move data. In this case, we will be moving data into the encrypted file system (encrypting it).
>
> The decrypt parameter is also a valid option here as well, which will produce the opposite effect. Decrypting data will be covered in a later section.
>
> *Note, all zNcrypt 3.3 encryption commands, by default, require free space equal to twice the size of the encrypted data. If your environment is lacking in free space, add '--per-file' to the end of the command which will move each file individually. Per file encryption only requires twice the size of the largest individual file.*

@<category>

> This is the access category that will be applied to the data being encrypted. When moving data into the encrypted file system, you will be protecting it with process-based access controls that will restrict access to only the processes that you allow. The naming convention of the category is entirely up to you (the '@' is required), but it is typically a good idea to keep it simple and memorable. Depending on what data you are encrypting, it is usually best to pick a name referencing the data encrypted. For example, a @mongodb category would be fitting for a MongoDB deployment (substitute in @mysql, @postgresql, etc to fit your environment).

<directory to encrypt>

This is the data that you would like to encrypt. This can be anything from a single file to a whole tree of directories. Just remember that the zNcrypt process starts after the system boots, so it's best not to encrypt system-required files/directories (the root partition, the entire /var directory, etc). A good example of what to encrypt could be /var/lib/mysql/data, /db/data, etc.

<encrypted mount-point>

The last parameter is where you want the data to be stored. This would be the path to the mount-point specified during the zncrypt-prepare command. In the example from the previous section above, this would be /var/lib/zncrypt/mount.

Once executed, the zncrypt-move command will move all of the data stored in the target directory into the encrypted storage directory, leaving a symbolic link directing processes to its new location.

To get a better idea of what this command did, you can run:

$ ls -l <directory to encrypt>

$ du -h <directory encrypted storage directory>

The output of which should demonstrate how the file-system is now laid out. Everything that was once in the target directory is now securely stored inside of the encrypted file-system, fully encrypted, and protected from any outside access.

Allowing Access to the Data

If you have already attempted to traverse the now-encrypted directory tree, you might have noticed that you're not able to (even as root). This is because zNcrypt acts as a whitelist (blocking everything unless you specify otherwise) for processes that you want to allow access. Since we have not added any new rules for processes nothing has access (probably not your intended use-case).

To add a new rule that would allow access to your desired process you'll need to use the zncrypt acl command:

$ sudo zncrypt acl --add --rule="ALLOW <category> * <path to binary file>"

The rule is pretty self explanatory, but let's go over each parameter:

zncrypt acl

This signifies that we will be doing something with our ACL set. The ACL set is the list of rules that govern access to the encrypted data. Adding/removing rules is typically the main use for this command, but for the full functionality check out the zncrypt manual page ($ man zncrypt).

--add

As mysterious as this parameter looks, this signifies that we will be adding a rule to our ACL set. The opposite command would be --del (which will be covered later).

--rule="ALLOW @<category> * <path to binary>"

This is the rule that we will be adding to our ACL set. To break this down even further, let's look at each part of the rule:

ALLOW

We will be allowing access to the encrypted data. The opposite would be DENY.

@<category>

Our user-defined category name that we encrypted the data with. This means that all data that was encrypted with the @mysql, @postgresql, @mongodb, @etc category would be affected by this rule.

*

This specifies the permissions for a specific path. This is useful if several different processes require access to the same directory tree. With this you can restrict which processes have access to which data.

For example, if you also need the /usr/bin/du binary to have access to a directory in your encrypted directory tree, you can specify /path/to/data/dustuff/* to ensure /usr/bin/du doesn't have access to the whole data set.

<path to binary>

This is the binary which we are allowing to access the data. If the program we are allowing access for has a binary then this is sufficient.

***Note**, zNcrypt fingerprints ACL binaries to ensure malicious tampering does not occur. This means that any update or change to an ACL binary will cause the binary to lose access to the encrypted data. To learn more about this fingerprinting process, please see the "**Updating ACL Fingerprints**" section below.*

Once the ACL rules have been added for the appropriate processes, you should be ready to restart your services.

Maintenance

Decrypting Data

Okay, you've tested it out and, frankly, it blew your mind, but now you want to get everything back to the way it was. You can decrypt your currently encrypted data with the following command:

```
$ sudo zncrypt-move decrypt <path to encrypted data>
```

Where the path to the encrypted data refers to the location of the symbolic link that was created during the 'zncrypt-move encrypt' command above.

Access Modes

zNcrypt offers three different 'modes' of access: enforcing, permissive, and admin.

enforcing - This mode strictly enforces all zNcrypt ACL rule sets, while blocking all other access attempts (which are then logged to dmesg). By default, a newly installed/restarted zNcrypt client will be in enforcing mode.

permissive - This mode allows full access to all encrypted data, regardless of whether or not ACL rules have been set. Something to note about this mode is that all interaction to encrypted data is logged as if it were blocked.

***Note**, that running in permissive mode for long periods of time will cause performance degradation.*

admin - This mode is the same as permissive mode, except without the extensive access logging.

To set zNcrypt into a specific access mode, run:

```
$ zncrypt set --mode=[enforcing|permissive|admin]
```

And to check which mode zNcrypt is currently in, you can run:

```
$ zncrypt status -d
```

Changing the zNcrypt Master Password

If you need to change the master password, execute the following command:

```
$ sudo zncrypt key --change
```

Removing ACL Rules

You can remove ACL rules just as easily as you added them. To display all currently-active rules, run:

```
$ zncrypt acl --list
```

Once you see the rule on the listing that you want to remove, you can delete it with:

```
$ zncrypt acl --del -n <ACL rule number>
```

Updating ACL Fingerprints

zNcrypt comes prepackaged with a security feature that prevents tampering with ACL binaries. When you added the ACL rules in the above section, zNcrypt collected a fingerprint of the binary from the initial rule creation. This fingerprint is compared to the current fingerprint every time the process attempts to access your encrypted data. If the binary changes (as it does with a system or process upgrade) the fingerprint won't match and the process will not be able to access the data.

This typically happens due to a system update that was unplanned, but could be potentially malicious in nature (which is why this feature exists). To see if your fingerprint has changed, execute:

```
$ zncrypt acl --list
```

If any ACL rules have a '!!' listed next to them, the fingerprint has changed. If the the binary change was a planned and expected change, run the following command to update the process fingerprint:

```
$ zncrypt acl --update
```

This command will prompt you for you master password, ensuring (hopefully) that you are an authorized user. Once complete, the ACL rules will be updated and access to the encrypted data will be restored.

Troubleshooting

If you continually run into any issues, please feel free to send us a quick note at support@gazzang.com.

Encryption Status

To check to see if the encrypted volumes are still mounted correctly, you can run:

```
$ sudo service zncrypt-mount status
```

Which will show you statistics around each encrypted partition. In the case that the encrypted partitions are unmounted or giving errors, you can also run:

```
$ sudo service zncrypt-mount [start|stop|restart]
```

The System Message Buffer

To verify your ACL's are configured correctly, you can execute:

```
$ dmesg
```

Which will output the system message buffer. This is the only location that zNcrypt will log permission issues.

Log Files

Another good place to check is the zNcrypt log directory. By default, zNcrypt logs will be located in the directory /var/log/zncrypt/.

Looking for More?

For advanced user commands and setup, see the full zNcrypt User Guide.

Support

Issues? Comments? Questions? Just want to say hi?

Feel free to contact our awesome support team at support@gazzang.com if you need any help with anything.

