Exercise 3 - Install Encrypt on a DataNode

Step 1 – Installing Encrypt

In order to install the zNcrypt utility, you will need to install all of the necessary system and package dependencies. If you want to go ahead and jump in, you can run the following command from the Linux CLI on a datanode in your class cluster:

```
curl -sL https://archive.gazzang.com/deployment/zncrypt-install.sh | sudo bash
```

Step 2 – Register with a Key Server

Once zNcrypt is fully installed, it is time to register with a zTrustee Server to store encryption keys. During this process, you will need to specify a "master passphrase" that will be used to lock the zNcrypt configuration. Ideally the master passphrase will be unique to the system, and separate from the Linux system administrator's password.

Note! If you are going to be using the bash scripts to aid in the configuration process, a master passphrase will be auto-generated for you. Once the configuration is completed, make sure you change it (or at least remove it from the file system).

To get started, run the following in your terminal:

```
sudo zncrypt register -s <hostname of key server> -o <organization name> --auth= <authorization code>
```

Where:

- *<hostname of the key server>* is the fqdn of the zTrustee Key Server you want to register against.
  Ex: `ztdemo.gazzang.net.` **`Use the hostname from Exercise 1.`**
- *<organization name>* is the name of the organization you would like to register the client against. Use the credentials from Exercise 1.
- *<authorization code>* is the authorization code/secret associated with the organization above. **Use the credentials from Exercise 1.**

cloudera
Ask Bigger Questions

Step 3 – ACLs in Hadoop

For Hadoop, setting ACL rules becomes slightly more complicated. The reason for this is because, since Hadoop is a `java`-based application, all Hadoop processes on a system (and all `java` processes, for that matter) will share the same `java` binary. This means that simply adding an ACL rule `java` will only keep out non-`java` processes, but all other `java` processes will still have access to the encrypted data (typically an undesirable feature).

To bypass this issue, zNcrypt has a secondary ACL attribute called a 'profile.' An ACL profile will allow you to further granulize access provisioning to the encrypted data even further by specifying `UID`, `GID`, and other CLI parameters. It is the zNcrypt profile feature that allows us to protect Hadoop and any HDFS data that we want protected.

A typical profile will look something like this:

```
{
"uid":"496",
"comm":"java",
"cmdline":"/usr/lib/jvm/jre-openjdk/bin/java -Dproc_datanode -Xmx1000m -
Dhadoop.log.dir=/var/log/hadoop-hdfs -Dhadoop.log.file=hadoop-hdfs-datanode-vagrant-
centos64.vagrantup.com.log -Dhadoop.home.dir=/usr/lib/hadoop -Dhadoop.id.str=hdfs -
Dhadoop.root.logger=INFO,RFA -Djava.library.path=/usr/lib/hadoop/lib/native -Dhadoop.policy.file=hadoop-
policy.xml -Djava.net.preferIPv4Stack=true -server -Dhadoop.security.logger=INFO,RFAS
org.apache.hadoop.hdfs.server.datanode.DataNode"
}
```

To get started, you will need to capture the profile of a running Hadoop system. If the datanode process is stopped on your machine, then you will need to start it now.

To collect the profile, we need to know the PID of the process we want to profile. In this case, let's use the datanode process:

```
sudo -s

# Capture PID(s) of datanode process. Replace [d]atanode below with whatever you want to capture ([h]adoop,
etc.)
PIDS="$(ps -ef | grep -i [d]atanode | awk '{ print $2}')"

# Generate profile(s), where we use a for loop just in case we get more than one result
for pid in ${PIDS[@]}; do zncrypt-profile --pid=$pid > /tmp/znc_profile_$pid; done

# Attach the profiles to the ACL's
for profile in "$(ls /tmp/znc_profile_*)"; do zncrypt acl --add -r "ALLOW @hadoop * $(grep "cmdline"
$profile | tr [=\"=],[=:=] " " | awk '{ print $2 }' | xargs readlink -f)" --profile-file=$profile; done
```


```
# If you added ACLs prior to attaching profiles, make sure you remove them once completed.
exit
```

Once all of the profiles have been added, you can list out the full set of ACL's by executing:

```
sudo zncrypt acl --list --all
```

Which should successfully list out the full profile for each ACL.

Note, you should repeat the above for Impala or any other processes that you want to allow access to encrypted data.

Step 4 – Preparing an encrypted store

Once the ACL's have been added, we'll want to prepare an encrypted storage area.

Command Overview

To encrypt any data, we will need to use the `zncrypt-prepare` command. This command has 4 parameters:

```
zncrypt-prepare [OPTIONS] <directory>  <mount-point> [-]
```

- The **directory** that the encrypted data will be stored. This can also be a block device
- The **mount-point** to use to move data to the encrypted store.

As an example:

```
sudo mkdir /var/lib/zncrypt/storage
sudo mkdir /var/lib/zncrypt/mount
sudo zncrypt-prepare /var/lib/zncrypt/storage /var/lib/zncrypt/mount
```

Step 5 - Encrypting Data (and, more importantly, Hadoop)

Once the ACL's have been added and a encrypted store created, we'll want to encrypt some data.

Note! Before encrypting ***ANYTHING***, you will want to make sure and **stop** all services that might need to access the target data. In the case of Hadoop, you will need to stop (at least) the DataNode process on each machine prior to encryption. Accessing data while it is being encrypted can lead to data corruption or loss of data.

Command Overview

To encrypt any data, we will need to use the `zncrypt-move` command. This command has 4 parameters:


```
zncrypt-move [action] @[category] [target] [mount]
```

- The **action** to complete (`encrypt` or `decrypt`)
- The **category** to associate for the data (in the example above, this was `@hadoop`
- The **target** data that we would like to encrypt (in the case of Hadoop, typically a HDFS data directory)
- The **mount** location where we would like to store the encrypted data (the encrypted mount-point from the `zncrypt-prepare` command above). ***Note!*** Be sure to keep size and space requirements in mind. For example, make sure the '`/data/N/dfs`' data is being stored on the '`/data/N/encrypted/mnt`' partition (for all data directories).

As an example:

Note! Before encrypting ***ANYTHING***, you will want to make sure and **stop** all services that might need to access the target data. In the case of Hadoop, you will need to stop (at least) the DataNode process on each machine prior to encryption. Accessing data while it is being encrypted can lead to data corruption or loss of data.

```
sudo zncrypt-move encrypt @hadoop /data1/dfs/dn /var/lib/zncrypt/mount
```

Where we are encrypting the '`/dfs/dn`' directory against the '`@hadoop`' ACL set, and the data will be stored inside of the '`/var/lib/zncrypt/mount`' encrypted partition. Once this command is executed, all of the data inside of the '`/dfs/dn`' directory will be moved into the '`/var/lib/zncrypt/mount`' partition (with a symbolic link pointing to the new location).

Note! Typically the deciding factor for how long an install will take depends on the amount of data that you will need to encrypt. As a rule of thumb, we like to say that the encryption process will take around the same amount of time as it would take to copy the data. Make sure you keep this in mind, especially when encrypting large datasets from a remote terminal (try to use `screen` or `nohup` when possible).

Verifying install

Once encryption is complete, restart the data node and perform hadoop fs operations.

```
sudo -u hdfs hadoop fs mkdir /users
sudo -u hdfs hadoop fs –mkdir /users/ec2-user
sudo -u hdfs hadoop fs –ls /users/ec2-user
sudo -u hdfs hadoop fs –ls /users/ec2-user
sudo -u hdfs hadoop fs –put /etc/hosts /users/ec2-user
sudo -u hdfs hadoop fs –get /users/ec2-user/hosts
```


Support

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

Feel free to contact our awesome support team at support@gazzang.com.

