Understanding Linux Kerberos Issues

Scope

This document provides information about using Kerberos on the Unified Linux Systems managed by the School of Informatics and Computing. There are many other standalone Linux systems in the school that are not part of the unified configuration and, therefore, this document does not apply to accounts on those systems. If you have any questions about Linux systems within the school, please contact us.

Background

The SoIC unified Linux systems have a system whereby you have a common home directory across all systems. The mechanism used to deliver your networked home directory to individual systems depends on the system type, as follows:

- **Servers** - A server is defined as a system that lives in the central SoIC machine room and is on a secure, private NFS network. For these systems, home directories are delivered using NFSv3. On these servers, it should be virtually transparent to the users that their home directory is not on the local disks on the server and these system are not subject to problems with kerberos authentication.

- **Workstations** - A workstation is defined as a system that is outside of the machine room or is not on the secure, private NFS network. These systems use kerberos authentication with NFSv4 to securely deliver user's home directories.

- **Hybrid** - There is also a hybrid configuration where a standard workstation is then configured with local home directories. Since the hybrid system has local home directories, that avoids the kerberos issues but also results in the loss of automated nightly backups and of a standard home directory across all systems.

The rest of this document deals with issues users may encounter on the Linux Workstations using kerberos with NFSv4.

Basic Workstation Usage Notes

In most cases, users of the unified Linux workstations will not encounter any problems as a result of the fact that kerberos authentication is used for home directory delivery. When you first log in, you get authenticated with kerberos and you gain access to your home directory for a period of 1 week. Every time you re-enter your password (eg. you unlock the screen) the counter is reset and you have another week of use. So, unless you leave the system locked for longer than 1 week you should not have any trouble accessing your home directory. However, there are a couple cases where remote use of the system may result in problems as a result of the way kerberos and NFSv4 authentication caching interact. The following sections describe the most common problems you are likely to encounter and include suggestions for how to avoid these problems.

The Easy Fix

Before we present the possible problems and solutions, it should be noted that you can avoid these problems entirely if you do your remote processing on a server instead of a workstation. For a list of the available servers, please see the KB document What Linux systems are available?. However, there are many cases where remote use of a workstation is required so the rest of this document will assist you with use of such systems.

**Problem 1: Authentication Expiration**

When you ssh into a workstation, you will be prompted to enter your password which will result in you authenticating with Kerberos. You can run the `klist` command after you log in to display your kerberos tickets and cache information. We have a mechanism in place that automatically renews your kerberos tickets but this automatic process can only renew for a period up to 1 week. So, if you ssh in and don't do anything to renew your kerberos credentials they will expire after 1 week. Once your credentials expire, you will no longer have access to your home directory. To avoid this problem, you can run `kinit` at any time to re-authenticate, thereby giving yourself another week of access.

If you have multiple ssh logins to a single workstation, the system will allow access to all of the sessions if any one of them has valid kerberos credentials. But, just keep in mind that if you log out of the one session that has valid kerberos credentials, the other logins will lose access and you will need to run `kinit` to renew your credentials in the other login session.

**Problem 2: Running Cron Jobs**
We have crontab disabled on the unified linux workstations so attempting to set this up will result in the following error:

```
$ crontab -e
You (username) are not allowed to use this program (crontab)
See crontab(1) for more information
```

The problem is that cron jobs run in an unauthenticated environment so they will not have permission to access your home directory. One workaround is to run your cron jobs on a server instead. However, if this isn't feasible because your cron job needs to perform a task on the workstation itself (and not just on your home directory) then we'll have to work with you on a solution. We have several options, including the creation of a local account on the workstation and allowing you to run cron jobs as this local user. Please contact us if you need to do this.

Problem 3: Running Persistent Remote Processes

Let's say you want to ssh into a workstation (and NOT a server like silo or tank where this is not a problem) and start a process that will keep running after you log out or you simply want to keep a persistent ssh login session open using screen. Here is what you may try to do that will cause problems:

**The Wrong Way**

```
host1$ ssh host2
username@host2's password: <you enter your password>
host2$ some_process &
host2$ logout
host1$
```

So, in this example you ssh into \texttt{host2} start up a process, and then logout. The problem with this is that your kerberos authentication is destroyed when you log out. The system will cache your authentication for a short time but within about 30 minutes, the process you have running will no longer have access to your home directory which will generally cause problems for that process. Even if that process isn't actively reading or writing files in your home directory, it can still cause the process problems if \$\text{HOME}\$ is pulled out from under it. So, if using a server (eg. silo, tank, hulk) is not an option, the solution is to use \texttt{screen} to give your process a session whereby the kerberos authentication survives. Here is the procedure using \texttt{screen} that replaces the above:
The Right Way

STARTING YOUR PROCESS
---------------------
host1$ ssh host2
username@host2's password: <you enter your password>
host2$ screen
host2$ module load screen-kinit
Password for username@ADS.IU.EDU: <you enter your password>
host2$ some_process &
host2$ CTRL-a CTRL-d
[detached]
host2$ logout
host1$

RENEWING YOUR CREDENTIALS (IF YOU NEED MORE THAN THE DEFAULT OF 1 WEEK)
-----------------------------------------------------------------------
host1$ ssh host2
username@host2's password: <you enter your password>
host2$ screen
host2$ kinit
Password for username@ADS.IU.EDU: <you enter your password>
host2$ CTRL-a CTRL-d
[detached]
host2$ logout
host1$

KILLING YOUR SESSION WHEN COMPLETED
-----------------------------------
host1$ ssh host2
username@host2's password: <you enter your password>
host2$ screen
host2$ module unload screen-kinit
host2$ exit
[screen is terminating]
host2$ logout
host1$

In this example, you are starting a screen session and authenticating in that session before you log out. The beauty of this approach (in addition to the fact that your home directory doesn't disappear) is that you can log in at a later time and reattach to this same session by running "screen -r". See the KB document How do I use screen on the Linux systems? for more information about using screen.

It should be noted that this approach will only give you access to your home directory for 1 week since your credentials expire after a week. If you need the process to run longer than a week, you can always ssh back into the system, reattach to the screen session with "screen -r", run "kinit" again, and then detach from the session with CTRL-a CTRL-d to give yourself another week.

Problem 4: SSH Logins and Authorized Keys

If you are using the SSH authorized_keys feature to allow ssh logins without requiring you to enter your password each time, you can run into some strange behavior. Here is an example of what can happen to cause problems:
1. You ssh into a system, enter your passphrase, get authenticated against kerberos, and then get a kerberos nfs ticket for access to your home directory and you are all good. At this point you can run `klist` and it shows your authentication ticket for the home directory server.

2. You then ssh into this same system and you get in **without** entering a passphrase via ssh keys because the system already has your nfs ticket from the login in step #1. So far so good. BUT this second shell is piggybacking on the original login and you don't have a ticket cache in this second shell. Running `klist` here just says 'no cache'.

3. You then log out of the first login (from step 1) which had your kerberos authentication.

4. At this point, things will work for a while in the second shell because the kernel caches your kerberos authentication for a short time. But, within 30 minutes or so the cached credentials will expire and your second login will lose access to your home directory.

5. You then ssh into the system again, it then prompts for your password, you get your kerberos credentials again, and you get in.

6. At this point, the second login again has access to your home directory because of this new login.

There are other scenarios that can cause similar odd behavior but they typically involve password-less logins using ssh keys.

There are a couple ways you can avoid this type of behavior but the simplest way is just to make sure your original ssh login (where you entered your passphrase) is not exited, thereby destroying the kerberos credentials. One good way to do this is to use screen for your initial login session as described in the previous section. By using screen, this initial login session becomes persistent and you can always re-attach if you need to reauthorize to give yourself another week.