Kerberos & SLURM using Auks

- Tutorial -
Agenda

Understanding the concepts

- Kerberos principles
- Auks principles
- Auks SLURM Spank plugin principles

Applying the concepts

- Kerberos infrastructure setup
- Auks infrastructure setup
- SLURM configuration for kerberos support with Auks
Understanding the concepts
- kerberos principles
Kerberos principles

Kerberos V5 : description

- Network authentication protocol for un-trusted environments
  - Authentication of peers in un-trusted environments

- Provides mutual authentication
  - Authenticates both sides of a communication

- Based on a trusted third party per Realm
  - Key Distribution Center (KDC)

- Based on symmetric encryption (in regular usage)
  - Shared secrets between the KDC and the different trusted parties

- Provide Single Sign-On in un-trusted environments
  - One initial credential acquisition enable automatic access to kerberized services

- Provide federated authentication
  - Through cross-realm authentications
Kerberos principles

Kerberos V5: specificities

- Expose key material with limited availability period
  - Relies on short lived credential

- Requires synchronized times between all peers
  - Validity period of credentials, replay attacks detection, ...

- Requires properly defined DNS configuration
  - Kerberos service « principals » (names) guessed from DNS reversed IP entries
  - Credentials associated to IP addresses according to DNS direct entries
Kerberos principles

Kerberos V5: terms and components

- **Realm**
  - Federation of peers and a trusted third party sharing a cryptographic secret with each of them

- **Principal**
  - Peer identity inside a realm

- **Authentication Server (AS)** *(Hosted by the KDC)*
  - Authenticate kerberos principals
  - Responsible for the AS_REQ/AS_REP kerberos exchange
    - TGT (Ticket Granting Ticket) delivery: short lived credentials for SSO

- **Service Server (SS)** *(Hosted by the KDC)*
  - Provides service tickets to authenticated peers
    - Those providing a valid TGT
  - Responsible for the TGS_REQ/TGS_REP kerberos exchange
    - TGS (Ticket Granting Service) delivery: short lived credentials to access services
Kerberos principles

Kerberos V5: advanced ticket features

- **Forwardable TGT**
  - TGT enabling to acquire new TGTs
  - Enable Single Sign-On (SSO)

- **Renewable tickets**
  - Distinction between *validity time* and *renewable time*
    - initial request time + validity time = expiration time
    - validity time lower or equal to renewable time
    - Exp: 1 day ticket renewable a week
  - expiration time inherited by TGS and forwarded TGT
  - Enable to extend the availability period of a credential by steps
    - Using a particular request to the KDC before « expiration time »
    - Up to initial request time + renewable time
  - Ensure that a lost ticket retrieved after the expiration time is no longer usable

- **Adresseless tickets (TGT/TGS)**
  - Tickets are bound to requester IP addresses by default
    - Kerberos services do not always check that field
  - Adresseless tickets are not bound to any IP addresses
    - Useful when using kerberos behind a NAT
Kerberos principles

Kerberos V5 protocol overview

- **Authentication stage**
  - **Stage 1**
    - AS_REQ to request a TGT (with encrypted preauth data to prove the identity of the requester)
  - **Stage 2**
    - AS_REP to get a TGT encrypted with the client shared key

- **Service ticket request**
  - **Stage 3**
    - TGS_REQ to request a TGS for the specific service « Service B »
  - **Stage 4**
    - TGS_REP to get a TGS encrypted with the TGT session key

- **Application access request**
  - **Stage 5**
    - AP_REQ to request a secured channel with the service
  - **Stage 6**
    - AP_REP to finalize the secure channel establishment with mutual authentication

- **Credential forwarding (Optional, required for SSO)**
  - **Stage 7**
    - TGS_REQ to request a TGT valid on « Service B » host
  - **Stage 8**
    - TGS_REP to get the forwarded TGT encrypted with the TGT session key
  - **Stage 9**
    - Transfer of the forwarded TGT over the secure channel
Kerberos principles

Kerberos V5 ecosystem

- Available for different platforms with different implementations
  - UNIX, Linux-based systems: MIT, Heimdal
  - Windows: Windows Active Directory implements a kerberos like/compatible protocol
  - ...

- Local area network as the primary target
  - But larger setup exist

- Multiple kerberized services available on Linux
  - Exp: OpenSSH, LDAP, ...

- Multiple kerberized Distributed File System
  - Exp: OpenAFS, NFSv3, NFSv4, NFSv4.x, Lustre!
Kerberos principles

Kerberos V5 interests in HPC environments

- Ease users connections to compute services
  - Workstation to login node seamless access

- Ease user connections from login node to allocated compute nodes
  - Monitoring
  - Debugging
  - Exotic MPI launcher
  - ...

- Secured access to provided services
  - Data staging (outside and/or inside the clusters)
  - Remote connections
  - ...
Kerberos principles

Kerberos V5 concerns in HPC environments

- Credential life cycle management
  - Defining the common session time for a HPC site
  - Ensuring the renewal of tickets in time

- Batch mode
  - Ensuring kerberized executions when no user directly involved

- Scalability
  - Trusted third party behavior with thousands of active nodes
  - Credential forwarding strategies with thousands of peers

- Integration
  - Ensuring kerberos support in HPC software stacks
Understanding the concepts

- Auks principles
Auks main objectives

- Provide kerberos credential support in non-interactive environments
  - Batch environment, CRON, ...

- Provide kerberos credential support for HPC environments
  - Large installations with large number of components
  - Ease Auks integration in Batch systems
    - Built-in support for SLURM through a dedicated SPANK plugin
      - Primary target
    - Command line client for other batch systems or usages
      - Exp : AUKS/GridEngine integration in Stanford FarmShare
Auks principles

Auks: description

- Distributed credential delegation system
  - Remote cache of kerberos credentials
    - Used to push or pull credentials
  - Regular renewal of cached tickets
    - for end-to-end life cycle usability of tickets
- Client/Server application
  - Kerberized Service to ensure the authentication and privacy of the exchanges

- External Linux tool
  - Easily integrated in external Apps (C-API, Command line clients)
  - Developed and tested on CentOS, RedHat, Fedora

- Not an authentication/authorization system

- OpenSource project
  - [https://sourceforge.net/projects/auks/](https://sourceforge.net/projects/auks/)
Auks principles

Auks overview

- Auksd, the central daemon
  - Multi-thread server written in C
  - Kerberized service
    - Require a kerberos keytab
    - NAT awareness (if configured)
  - Authenticate client requests
    - Using associated kerberos principal
  - Serve add/get/remove/dump TGT requests
    - After authorization using Auks's ACL
  - Stores users TGT in a FS directory
    - for persistence
    - At most one TGT per user
    - Principal/uid mapping on server side
  - Cache TGT in memory
    - to speed-up retrieval
  - Get requests providing **uid**
    - Avoid to guess the principal to request when used with cross-realm
  - HA using an external tool
    - No internal mechanism yet
Auks principles

Auks overview

- **Libauksapi**
  - Automatic switch to backup server in case of failure
  - Configurable timeouts, retry delays and retries amount
  - Automatically get addressless TGTs to process add requests
    - Cached tickets usable from any hosts

- **Auks**
  - Simple program wrapping and configuring the API calls
  - Require a valid TGT in client environment to succeed

- **Auksdrenewer**
  - Implementation of the Auksd renewal mechanism
  - Externalized due to thread safety issues in initial kerberos libraries used
  - Allowed to dump auksd TGT cache
Auks principles

Auksd internals

- Credential repository (memory cache)
- Cleaner
- Worker
- Worker
- Worker

TCP/IP stack

Socket listener

Socket queue
Auks principles

Auks authorization

- Authorization rules defined by ACL
  - Based on
    - Requester kerberos principal
    - Requester host
  - Determine requesters role/privilege
    - Guest: add request for personal credential only
    - User: add/get/remove requests for personal credential only
    - Admin: add/get/remove/dump requests for any credential

Auks features

- Scalability
  - Single addressless ticket managed per user
  - Retrieved by each involved compute node using a highly parallel server
    - Several thousands « get » requests per second
- Everlasting jobs
  - Using Auks ticket renewal mechanism (new get to the Auksd, instead of KDC req)
  - Long running jobs as long as users refresh the credential stored in auks
Auks principles

Auks usage example scenario

- Users add their credentials to the remote cache
- Users request their batch execution
- Auksdrenewer/Auksd renew the credentials waiting for the associated executions
- Batch system with admin privilege retrieve the credential using Auksd

- Batch system set the credential in the user env and start the execution
- The batch execution can renew the credential using Auksd (if user privilege)
  - Avoid to contact the external KDC with massively parallel/simultaneous requests
Understanding the concepts

- Auks Spank plugin principles
Auks SLURM Spank plugin principles

Auks SPANK plugin description

- **SLURM SPANK API**
  - Provides hooks to perform various actions at different stages of job/step life cycles

- **Auks client embedded in a SPANK plugin for SLURM**
  - **User side**
    - auks « user » privilege required
    - « Add » request at submission time
    - « Get » requests during execution
  - **Slurmd side**
    - Auks « admin » privilege required
    - « Get » requests at the very beginning of execution
      - to set up a kerberized env for the user
  - **1/N strategy**
    - 1 « add » request per submission/allocation
    - N « get » requests, 1 per allocated node
  - **Stackable**
    - Consecutive SPANK plugins can reuse the kerberos ticket
      - Slurm-spank-X11 for internal ssh connections
        - https://github.com/hautreux/slurm-spank-x11
Auks SPANK plugin : srun scenario
Auks SLURM Spank plugin principles

Auks SPANK plugin: sbatch scenario

ACLs:
*@MYREALM from anywhere is User
slurm/*@MYREALM from anywhere is Admin
Auks SLURM Spank plugin principles

Feedback

- Used on Tera100 and PRACE Curie machine
  - ~4k nodes and ~5.5k nodes each
  - 1 single auksd server with 1000 worker threads per cluster
    - Used for « add » / « intial get » / « renewal get » requests
  - In production since 2009
  - Current production version : 0.4.0

Usage

- Mostly for data staging in jobs (remote scp)
- Internal ssh connections (X11 tunnel setup with slurm-spank-x11)
- Not yet used for FS access
  - Planned for the near future
Limitations

- Every compute node has the auks « admin » privilege
  - allowed to acquire any ticket using « auks »
  - Would require major modifications in SLURM internals to change that

- Scalability issue with a very large amount of nodes
  - Not sure to manage properly several 10k « get » requests
  - Would require major modifications in SLURM internals to
    - Acquire one ticket at execution start
    - Propagate it securely to all the involved nodes (like sbcast)

- Scalability issue of Kerberos infrastructure when TGT are used in parallel by all the nodes
  - TGS request from every allocated nodes to access the kerberized FS
    - TGS prefetching and caching using Auks is planned to cope with that
  - Not only a AUKS SPANK plugin problem but a generic one with kerberos at scale
Applying the concepts
Applying the concepts

Assumptions

- RedHat like Linux system
  - If you are using something else, you are smart enough to know how to deal with the conversion :)

- MIT Kerberos implementation
  - Auks not supported/tested with Heimdal implementation
  - Basic/Default configuration will be used when possible
    - Kerberos is not the focus of this presentation

- Only a single kerberos Realm is used
  - Simpler to setup in a few minutes
  - Auks properly work with multiple realms and cross-realm relations

- A single laptop as the client, server, SLURM login and compute nodes
  - Sufficient to reproduce the classical scenarios
  - Avoid time synchronization and DNS issues
  - Everyone who want to try should be able to do it during this presentation
    - assuming an existing SLURM setup on your laptop... But that is certainly the case :)

Applying the concepts

- Kerberos infrastructure setup
Kerberos infrastructure setup

Installation

- RPM packages required
  - Client side: krb5-workstation
  - Server side: krb5-workstation, krb5-server
  - Development side (to compile Auks): krb5-devel

Configuration

- Configuration files
  - Server side: /var/kerberos/kdc.conf
  - Client side: /etc/krb5.conf

```bash
[root@leaf ~]# vim /var/kerberos/kdc.conf
[root@leaf ~]# cat /etc/krb5.conf
[logging]
default = FILE:/var/log/krb5libs.log
dc = FILE:/var/log/krb5kdc.log
admin_server = FILE:/var/log/kadmind.log
[libdefaults]
default_realm = TREE.ORG
dns_lookup_realm = false
dns_lookup_kdc = false
ticket_lifetime = 24h
renew_lifetime = 7d
forwardable = true
noaddresses = false
preferred_pcreauth_types = "17, 16 15, 14"
[realms]
TREE.ORG = {
acl_file = /var/kerberos/krb5kdc/kadm5.acl
dict_file = /usr/share/dict/words
admin_keytab = /var/kerberos/krb5kdc/kadm5.keytab
supported_enctypes = aes256-cts:normal aes128-cts:normal
max_renewable_life = 7d
}
[root@leaf ~]#
```

```bash
[root@leaf ~]# vim /var/kerberos/krb5kdc/kdc.conf
[root@leaf ~]# cat /var/kerberos/krb5kdc/kdc.conf
[kdcdefaults]
kdc_ports = 88
kdc_tcp_ports = 88

[realms]
TREE.ORG = {
acl_file = /var/kerberos/krb5kdc/kadm5.acl
dict_file = /usr/share/dict/words
admin_keytab = /var/kerberos/krb5kdc/kadm5.keytab
supported_enctypes = aes256-cts:normal aes128-cts:normal
max_renewable_life = 7d
}
[root@leaf ~]#
```
Kerberos infrastructure setup

Initialization

- Kerberos DB initialization
- Kerberos KDC initial start
- First principals creation
  - Including host keytab creation: `/etc/krb5.keytab`

```
[root@leaf ~]# kdb5_util create -s
Loading random data
Initializing database '/var/kerberos/krb5kdc/principal' for realm 'TREE.ORG',
master key name 'K/M@TREE.ORG'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key:
Re-enter KDC database master key to verify:
[root@leaf ~]# service krb5kdc start
Starting krb5kdc (via systemctl):                          [  OK  ]
[root@leaf ~]# kadmin.local
Authenticating as principal root/admin@TREE.ORG with password.
kadmin.local: ank -randkey +requires_preauth host/leaf.tree.org@TREE.ORG
WARNING: no policy specified for host/leaf.tree.org@TREE.ORG; defaulting to no policy
Principal "host/leaf.tree.org@TREE.ORG" created.
kadmin.local: ktadd -k /etc/krb5.keytab host/leaf.tree.org@TREE.ORG
Entry for principal host/leaf.tree.org@TREE.ORG with kvno 2, encryption type aes256-cts-hmac-sha1-96 added to keytab WRFILE:/etc/krb5.keytab.
Entry for principal host/leaf.tree.org@TREE.ORG with kvno 2, encryption type aes128-cts-hmac-sha1-96 added to keytab WRFILE:/etc/krb5.keytab.
kadmin.local: ank +requires_preauth mat
WARNING: no policy specified for mat@TREE.ORG; defaulting to no policy
Enter password for principal "mat@TREE.ORG":
Re-enter password for principal "mat@TREE.ORG":
Principal "mat@TREE.ORG" created.
kadmin.local: quit
[root@leaf ~]#
```
Kerberos infrastructure setup

Validation

- OpenSSH configuration
  - GSSAPI Authentication (the SSO part of the kerberos authentication in SSH)
    - With Credential delegation (For cascading SSO)
    - With Credential cleanup

- Client credential initialization

- OpenSSH connection
  - Using GSSAPI

```
[root@leaf ~]# vim /etc/ssh/sshd_config
[root@leaf ~]# vim /etc/ssh/ssh_config
[root@leaf ~]# grep GSSAPI /etc/ssh/sshd_config | egrep -v "^#"
GSSAPIAuthentication yes
GSSAPICleanupCredentials yes
[root@leaf ~]# grep GSSAPI /etc/ssh/sshd_config | egrep -v "^#"
GSSAPIAuthentication yes
GSSAPIDelegateCredentials yes
[root@leaf ~]# service sshd restart
Restarting sshd (via systemctl): [ OK ]
```

```
[mat@leaf ~]$
kinit
Password for mat@TREE.ORG:

[mat@leaf ~]$
klist
Ticket cache: FILE:/tmp/krb5cc_500
Default principal: mat@TREE.ORG

Valid starting     Expires            Service principal
09/21/12 17:12:21  09/22/12 17:12:19  krbtgt/TREE.ORG@TREE.ORG
renew until 09/28/12 17:12:21

[mat@leaf ~]$
ssh leaf klist
Ticket cache: FILE:/tmp/krb5cc_500_ktKqRf2319
Default principal: mat@TREE.ORG

Valid starting     Expires            Service principal
09/21/12 17:12:29  09/22/12 17:12:19  krbtgt/TREE.ORG@TREE.ORG
renew until 09/28/12 17:12:21

[mat@leaf ~]$
klist
Ticket cache: FILE:/tmp/krb5cc_500
Default principal: mat@TREE.ORG

Valid starting     Expires            Service principal
09/21/12 17:12:21  09/22/12 17:12:19  krbtgt/TREE.ORG@TREE.ORG
renew until 09/28/12 17:12:21
09/21/12 17:12:29  09/22/12 17:12:19  host/leaf.tree.org@TREE.ORG
renew until 09/28/12 17:12:21

[mat@leaf ~]$
```
Kerberos infrastructure setup

How it worked

Authentication stage

```
bash-3.2$ kinit
Password for mat@TREE.ORG:
bash-3.2$ klist
  Valid starting  Expires  Service principal
  09/21/12 17:12:21  09/22/12 17:12:19  krbtgt/TREE.ORG@TREE.ORG
  renew until 09/28/12 17:12:21
...
bash-3.2$
```

Service ticket request

- Not required in real scenario (automatic)

```
bash-3.2$ kvno host/leaf.tree.org
host/leaf.tree.org@TREE.ORG: kvno = 2
bash-3.2$ klist
  09/21/12 17:12:27  09/22/12 17:12:19  host/leaf.tree.org@TREE.ORG
  renew until 09/28/12 17:12:21
...
bash-3.2$
```

Application access request

```
bash-3.2$ ssh leaf.tree.org klist
...
  Valid starting  Expires  Service principal
  09/21/12 17:12:29  09/22/12 17:12:19  krbtgt/TREE.ORG@TREE.ORG
  renew until 09/28/12 17:12:21
...
bash-3.2$
```
Kerberos infrastructure setup

Other interesting commands

### TGT renewal

bash-3.2$ klist

```
... Valid starting          Expires                   Service principal
09/21/12 23:47:29  09/22/12 23:47:27  krbtgt/TREE.ORG@TREE.ORG
... bash-3.2$ kinit -R
bash-3.2$ klist
... Valid starting          Expires                   Service principal
09/21/12 23:47:35  09/22/12 23:47:33  krbtgt/TREE.ORG@TREE.ORG
... bash-3.2$
```

### Adreessless tickets retrieval

bash-3.2$ kinit
Password for mat@TREE.ORG:
bash-3.2$ klist -a

```
... Valid starting          Expires                   Service principal
09/21/12 23:49:23  09/22/12 23:49:20  krbtgt/TREE.ORG@TREE.ORG
... bash-3.2$ kinit -A
Password for mat@TREE.ORG:
bash-3.2$ klist -a
... Valid starting          Expires                   Service principal
09/21/12 23:50:34  09/22/12 23:50:32  krbtgt/TREE.ORG@TREE.ORG
... bash-3.2$
```
Applying the concepts

- Auks infrastructure setup
Auks infrastructure setup

Installation

  - Then build RPM packages (or use `./configure ; make ; make install`)

- RPM packages required
  - Client/Server side : auks
  - Login/compute nodes : auks, auks-slurm

```
[mat@leaf auks]$ rpmbuild -ta auks-0.4.0.tar.gz
...  
Wrote: /home/mat/rpmbuild/SRPMS/auks-0.4.0-1.src.rpm
Wrote: /home/mat/rpmbuild/RPMS/x86_64/auks-0.4.0-1.x86_64.rpm
Wrote: /home/mat/rpmbuild/RPMS/x86_64/auks-devel-0.4.0-1.x86_64.rpm
Wrote: /home/mat/rpmbuild/RPMS/x86_64/auks-slurm-0.4.0-1.x86_64.rpm
...

[root@leaf x86_64]# rpm -i auks-0.4.0-1.x86_64.rpm auks-slurm-0.4.0-1.x86_64.rpm
...  
[root@leaf ~]#
```
Configuration

- Client / Server side
  - Common configuration
  - Server configuration
    - `/etc/auks/auks.conf`

- Server only
  - ACL rules
    - Roles definition
    - `/etc/auks/auks.acl`

```bash
[root@leaf ~]# cp /etc/auks/auks.conf.example /etc/auks/auks.conf
[root@leaf ~]# emacs -nw /etc/auks/auks.conf
[root@leaf ~]# diff /etc/auks/auks.conf.example /etc/auks/auks.conf
5a6,16
> rule {
>   principal = ^host/leaf.tree.org@TREE.ORG$ ;
>   host = * ;
>   role = admin ;
> } > rule {
>   principal = ^[:alnum:]+@TREE.ORG$ ;
>   host = * ;
>   role = user ;
> } >
[root@leaf ~]#
```

```bash
[root@leaf ~]# cp /etc/auks/auks.acl.example /etc/auks/auks.acl
[root@leaf ~]# emacs -nw /etc/auks/auks.acl
[root@leaf ~]# diff /etc/auks/auks.acl.example /etc/auks/auks.acl
5a6,16
> rule {
>   principal = ^host/leaf.tree.org@TREE.ORG$ ;
>   host = * ;
>   role = admin ;
> } > rule {
>   principal = ^[:alnum:]+@TREE.ORG$ ;
>   host = * ;
>   role = user ;
> } >
[root@leaf ~]#
```
Auks infrastructure setup

Initialization & validation

- Auks service startup
  
  - Auks service test
    - As a user

  ```
  [root@leaf ~]# service auksd start
  Starting auksd (via systemctl): [ OK ]
  [root@leaf ~]# tail -f -n4 /var/log/auksd.log
  Sat Sep 22 23:06:12 2012 [INFO1] [euid=0,pid=7380] auksd : 11/11 workers launched
  Sat Sep 22 23:06:12 2012 [INFO1] [euid=0,pid=7380] dispatcher: auksd stream created on leaf.tree.org:12345 (fd is 1)
  Sat Sep 22 23:06:12 2012 [INFO1] [euid=0,pid=7380] dispatcher: socket 1 listening queue successfully specified
  Sat Sep 22 23:06:12 2012 [INFO2] [euid=0,pid=7380] worker[0] : auks cred repo cleaned in ~0s (0 creds removed)
  ```

  ```
  [mat@leaf auks]$ kinit
  Password for mat@TREE.ORG:
  [mat@leaf auks]$ auks -v
  Auks API request succeed
  [mat@leaf auks]$
  ```

  ```
  [root@leaf ~]# tail -n1 /var/log/auksd.log
  Sat Sep 22 23:06:47 2012 [INFO2] [euid=0,pid=7380] worker[7] : mat@TREE.ORG from 192.168.0.14 : ping request succeed
  ```

  ```
  [mat@leaf auks]$ auks -a
  Auks API request succeed
  [mat@leaf auks]$
  ```

  ```
  [root@leaf ~]# ls /var/cache/auks/
  aukscc_500
  [root@leaf ~]#
  ```
Applying the concepts

- SLURM configuration for kerberos support with Auks
Configuration & Initialization

Login/Compute nodes

- Auks SPANK plugin configuration
- `/etc/slurm/plugstack.conf.d/auks.conf`

```bash
[root@leaf ~]# cp /etc/slurm/plugstack.conf.d/auks.conf.example /etc/slurm/plugstack.conf.d/auks.conf
[root@leaf ~]# emacs -nw /etc/slurm/plugstack.conf.d/auks.conf
[root@leaf ~]# diff -u /etc/slurm/plugstack.conf.d/auks.conf.example /etc/slurm/plugstack.conf.d/auks.conf
--- /etc/slurm/plugstack.conf.d/auks.conf.example 2012-09-22 22:42:22.000000000 +0200
+++ /etc/slurm/plugstack.conf.d/auks.conf 2012-09-22 23:19:01.071361060 +0200
@@ -37,4 +37,4 @@
   # preventing the Auks plugin to put again the credential on the auks server.
   #
   #-------------------------------------------------------------
-#optional auks.so default=disabled spankstackcred=no minimum_uid=0
+optional auks.so default=disabled spankstackcred=no minimum_uid=500
[root@leaf ~]#

[root@leaf ~]# emacs -nw /etc/slurm/slurm.conf
[root@leaf ~]# grep -i plugstack /etc/slurm/slurm.conf
PlugStackConfig=/etc/slurm/plugstack.conf.d/auks.conf
[root@leaf ~]#
[root@leaf ~]# service slurm restart
Restarting slurm (via systemctl): [ OK ]
[root@leaf ~]#```
Validation

```bash
[mat@leaf auks]$ kinit
Password for mat@TREE.ORG:
[mat@leaf auks]$ srun --auks=yes klist
Ticket cache: FILE:/tmp/krb5cc_500_17_m1q54v
Default principal: mat@TREE.ORG

Valid starting   Expires       Service principal
09/22/12 23:21:48 09/23/12 23:06:41  krbtgt/TREE.ORG@TREE.ORG
    renew until 09/29/12 23:06:41
[mat@leaf auks]$
```

```
[root@leaf ~]# tail -n2 /var/log/auksd.log
```

```
[mat@leaf auks]$ srun --auks=done klist
Ticket cache: FILE:/tmp/krb5cc_500_18_VRhpzq
Default principal: mat@TREE.ORG

Valid starting   Expires       Service principal
09/22/12 23:21:48 09/23/12 23:06:41  krbtgt/TREE.ORG@TREE.ORG
    renew until 09/29/12 23:06:41
[mat@leaf auks]$
```

```
[root@leaf ~]# tail -n2 /var/log/auksd.log
```

SLURM configuration for kerberos support with Auks

Validation

```
$ kdestroy
$ klist
klist: No credentials cache found (ticket cache FILE:/tmp/krb5cc_500)
$
$ srun ssh -oBatchMode=yes leaf.tree.org hostname
Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).
srun: error: leaf10: task 0: Exited with exit code 255

$ srun --auks=done ssh -oBatchMode=yes leaf.tree.org hostname
leaf
$

$ srun --auks=done ssh -oBatchMode=yes leaf.tree.org klist
Ticket cache: FILE:/tmp/krb5cc_500_QdfxD10995
Default principal: mat@TREE.ORG

Valid starting   Expires   Service principal
09/23/12 00:06:57 09/23/12 23:06:41  krbtgt/TREE.ORG@TREE.ORG
renew until 09/29/12 23:06:41
```

[mat@leaf auks]$
Thank you for your attention

Questions?