Paper 3137-2019

SAS® Viya® and Kerberos: From the Ground Up

Michael Shealy and Spencer Hayes, Cached Consulting LLC

ABSTRACT

Kerberos is a network authentication protocol commonly found in medium and large enterprises. It uses strong cryptography to prove the identities of users, hosts, and services. SAS® has long been able to integrate with Kerberos to not only support its authentication strengths, but to enable user-friendly features such as Integrated Windows Authentication (IWA). However, SAS delegates some key configuration and operational tasks to the Kerberos realm and operating system layers, and their respective administrators. These include Kerberos principal configurations, ticket requests, ticket cache management, OS-layer authentication, and more. For this reason, it can be difficult for SAS administrators to fully see and understand the entire SAS and Kerberos integration picture.

We outline the configuration of a real-world proof-of-concept environment, which includes SAS® Viya® and Hadoop over CentOS, integrated with Microsoft Active Directory Kerberos authentication, and with support for IWA from the user's desktop. With a deep understanding of how this specific environment is configured for Kerberos integration from the OS layer up through the applications, SAS administrators should be able to better understand deployment and integration options in their own corporate environment. They should also be able to improve their troubleshooting capabilities in existing environments with Kerberos integrations.

Note that the key purpose of this paper is not to show each step of the kerberization process in explicit detail. Our environment consisted of CentOS 7.6, Cloudera 6.1.1, and SAS Viya 3.4, and if we took that approach this paper would quickly become stale. Instead our approach is to systematically walk through the key configurations that are common across all kerberized application deployments on Linux operating systems, using Hadoop and SAS Viya as examples, in an effort to keep this paper relevant for years to come.

TABLE OF CONTENTS

KERBEROS BASICS

OVERVIEW

Kerberos is a network authentication protocol that uses encrypted “tickets” to allow users and services to securely prove their identity to each other. We’ll explain the basics of how Kerberos uses these tickets to authenticate users in an example shortly, but first a quick history and a few important definitions.

HISTORY

Kerberos was first developed in the late 1980s at the Massachusetts Institute of Technology (MIT), and the "MIT Kerberos" distribution maintained by MIT remains a popular distribution in the marketplace. Beginning with the "Windows 2000" software, Microsoft made Kerberos its default authentication protocol, and now Microsoft's "Active Directory" (AD) Kerberos variant is likely the most popular implementation of the Kerberos protocol worldwide.

DEFINITIONS

KDC (Key Distribution Center): The KDC Server is the central "brain" of the Kerberos infrastructure, and maintains a database of user, host and service identities and their passwords. The KDC is composed of two components, an Authentication Server (AS) and a Ticket Granting Server (TGS), which often reside on the same host. A SAS Administrator normally will not need to worry about the different roles of the AS and TGS, and can simply refer to the system as the "KDC", but the two components are important for explaining a few of the core Kerberos communications we cover below.

Kerberos Realm: A Kerberos realm is a defined set of users, hosts and services (e.g. instances of SAS CAS, http, ssh, or Hadoop hdfs on a network) which are known and managed by an instance of a KDC. A realm name frequently looks like a DNS domain, but by convention it consists of uppercase letters. In our paper, our realm name is CCLLC.INTERNAL. Note that multiple KDC's in a Master/Secondary configuration may support a single Kerberos realm for availability and performance reasons, but they will operate with the replicated contents of the same Kerberos database.

Kerberos Principal: A Kerberos Principal is a user, host or service known to a Kerberos realm. The name of a Kerberos Principal consists of three parts: the primary, the instance and the realm in the format *primary/instance@REALM*.

- The *primary* can be any ascii string which can represent a user, host or service in a realm (e.g., "bill" for user Bill, "sascas" for Viya CAS, or "http" for Web Services).
- The *instance* is an optional ascii string that qualifies the primary, in order to ensure naming uniqueness across the realm. It is generally the fully-qualified domain name of a server or a DNS alias, but in MIT Kerberos the instance-string "admin" often defines the Kerberos administrator accounts as well.

Here are a few examples of principals we'll use later in this paper:

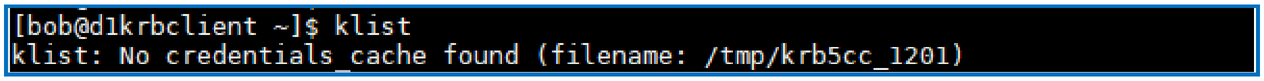
- bob@CCLLC.INTERNAL: The User Principal for person "bob" in the CCLLC.INTERNAL realm.
- d1viyasvclayer1$@CCLLC.INTERNAL: The "User-like" Principal for computer d1viyasvclayer1 (one of two Viya service-layer/programming hosts in our environment). These "machine account" principals are only used in AD, and we'll clarify "user-like" later in the paper.
- sascas/d1casprimary.ccllc.internal@CCLLC.INTERNAL: The Service Principal which supports the kerberized CAS service on our primary CAS server, d1casprimary.ccllc.internal.

KERBEROS AUTHENTICATION EXAMPLE

One of the strongest reasons to use Kerberos is because it does not send passwords over a network, where they are vulnerable to be stolen. Kerberos can do this by encrypting communications with secret keys *created* from a password, instead of sending the password itself to a server to authenticate. As long as both the user and the server know the user's password, they can send private messages to each other by creating a secret encryption key with that password via an agreed-upon algorithm.

An example will help explain how this works. Note that we have simplified the Kerberos communications in this example, to convey the key information necessary to understand the remainder of the paper. References containing more detailed explanations of the Kerberos communications are available at the end of the paper.

Imagine we have an environment where user “Bob” wants to access SAS Viya Visual Analytics in his browser, but SAS is using Kerberos security for authentication. How can Bob access the system? “klist” is the program used to show which Kerberos credentials a user has, and at the beginning of our story Bob has none (see Output 1).



```
[bob@d1krbclient ~]$ klist
klist: No credentials cache found (filename: /tmp/krb5cc_1201)
```

Output 1. Klist Display With No Available Credentials

Step 1

In order to acquire the Kerberos credentials he needs, Bob will first send a request to the KDC “Authentication Server” (see Figure 1), and ask for a “Ticket-Granting Ticket” (TGT). This TGT will be needed to talk to the KDC “Ticket Granting Server”, who will soon help facilitate access to the SAS Viya Visual Analytics application.

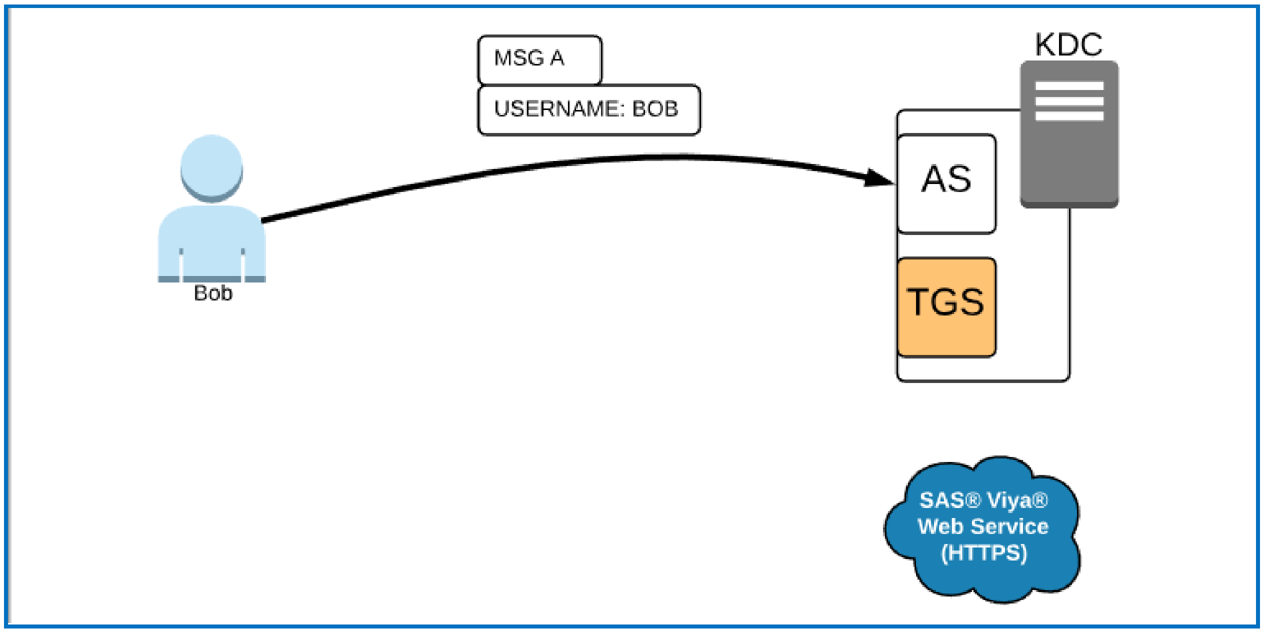
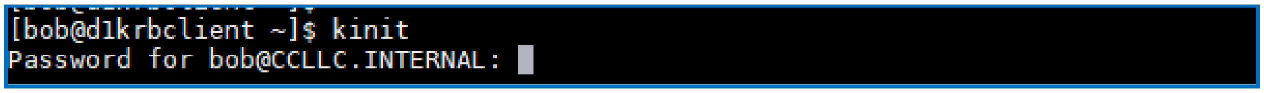


Figure 1. Initial User Request To The KDC Authentication Server

The process of acquiring a TGT can be started with the “kinit” command, which prompts Bob for a password, as shown in Output 2.



```
[bob@d1krbclient ~]$ kinit
Password for bob@CCLLC.INTERNAL: 
```

Output 2. Kinit Command Line Prompt For A Password

Step 2

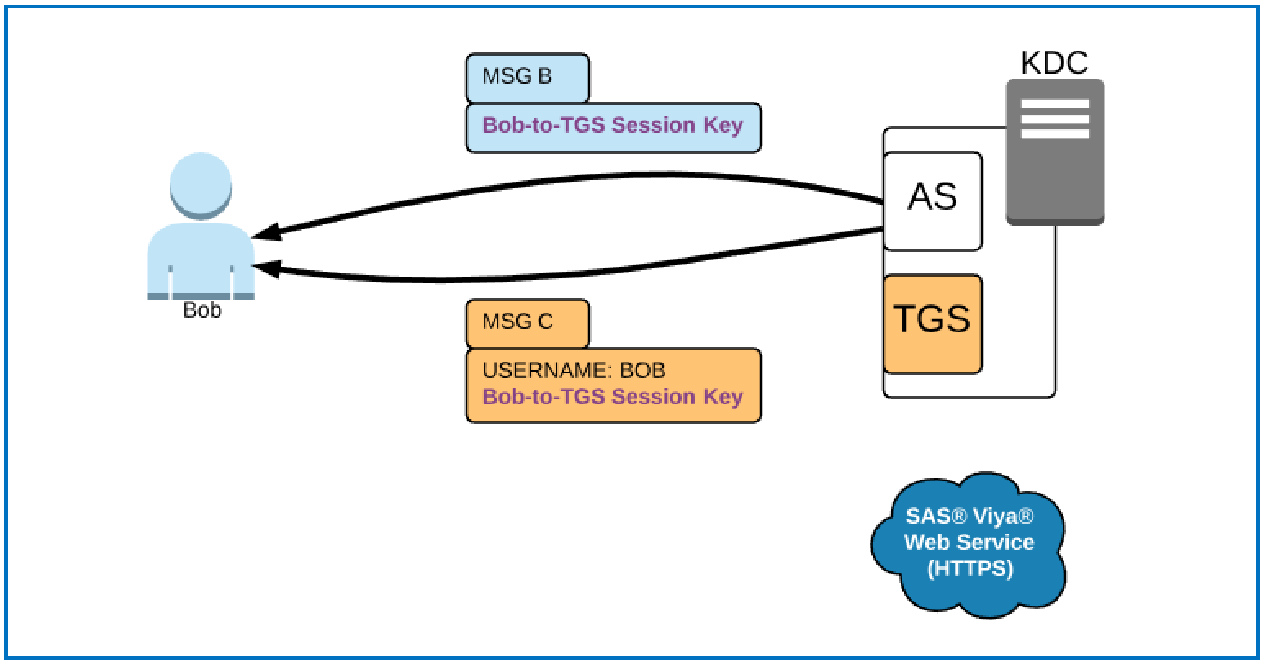
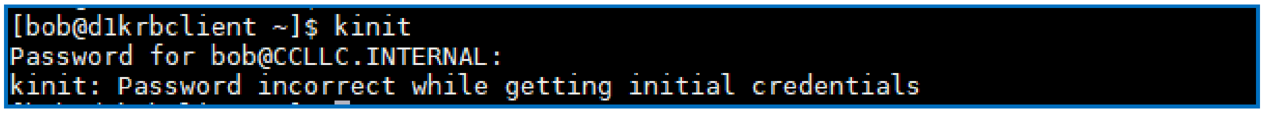


Figure 2. KDC Authentication Server Reply To User (With TGT)

The KDC Authentication Server will send Bob back two messages (see Figure 2), one which he can decrypt and one which he cannot.

Message B is encrypted with a secret key created with Bob's password (which the Authentication Server knows). This message will be decrypted with the password Bob typed in when he ran "kinit" on his computer. The message contains a second secret encryption key, which will be used to secure future communications between Bob and the Ticket Granting Server (the "Bob-to-TGS Session Key").

Note that Bob cannot acquire the soon-to-be-necessary second secret key unless he can decrypt Message B, and Message B can only be decrypted by Bob typing in his correct password…this is the fundamental way Kerberos can authenticate users without a password actually traversing the network. If Bob types in an incorrect password and Message B cannot be decrypted, he'll receive a message saying so (see Output 3).

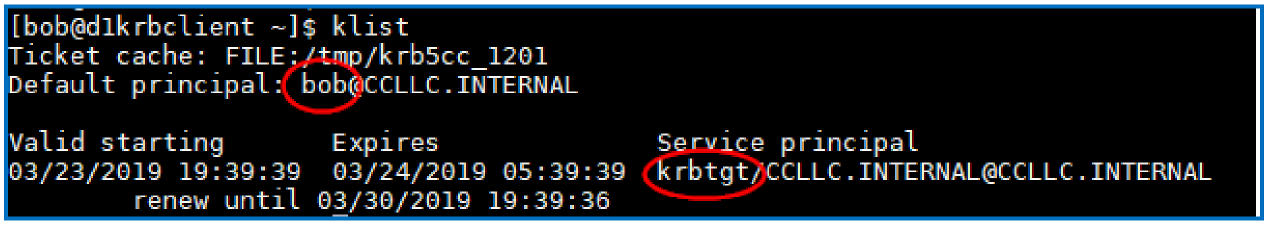


```
[bob@d1krbclient ~]$ kinit
Password for bob@CCLLC.INTERNAL:
kinit: Password incorrect while getting initial credentials
```

Output 3. Kinit Display With Incorrect Password Entry

Message C is encrypted with a secret key created with the password of the *Ticket Granting Server* (which the KDC Authentication Server knows, but Bob does not). The message contains Bob's username and a copy of the "Bob-to-TGS Session Key". It is usually referred to as the "Ticket Granting Ticket" (TGT) in Kerberos documentation.

When Bob enters his correct password, his klist output will look like Output 4.



```
[bob@d1krbclient ~]$ klist
Ticket cache: FILE:/tmp/krb5cc_1201
Default principal: bob@CCLLC.INTERNAL

Valid starting       Expires              Service principal
03/23/2019 19:39:39  03/24/2019 05:39:39  krbtgt/CCLLC.INTERNAL@CCLLC.INTERNAL
        renew until 03/30/2019 19:39:36
```

Output 4. Klist Display Showing Ticket Granting Ticket (TGT)

It shows that User Principal "bob" is in possession of a ticket associated with Service Principal "krbtgt", which is a Ticket Granting Ticket supporting communications with the Ticket Granting Server. The output also shows information such as the location where the ticket is stored, and its expiration date.

Step 3

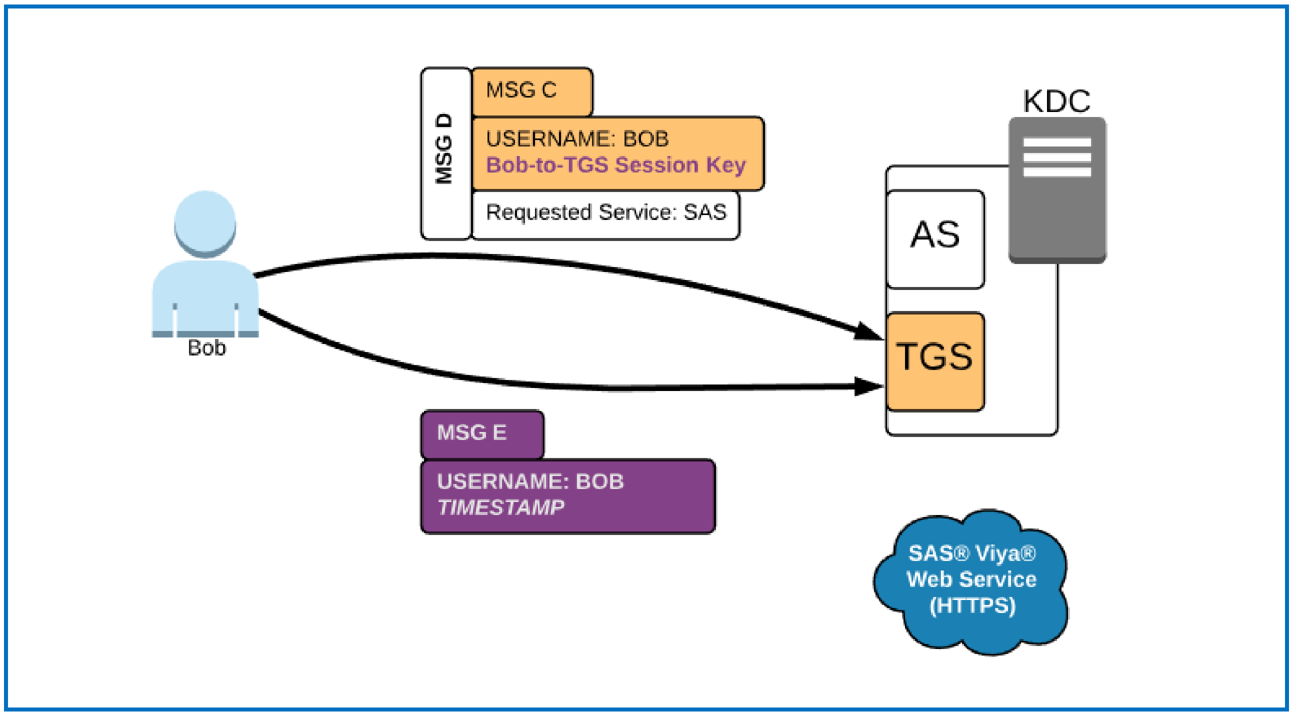


Figure 3. User Request To KDC Ticket Granting Server For A Service Ticket

Bob will then ask the Ticket Granting Server (TGS) for a ticket to talk to the Kerberos-secured SAS Visual Analytics web application. Bob sends two messages to the TGS (see Figure 3).

Message D consists of the TGT (which Bob could not decrypt, but the Ticket Granting Server can), along with the name of the application service that Bob wants to talk to (SAS).

Message E is called an "Authenticator", and it's essentially Bob's name and a timestamp, encrypted with the "Bob-to-TGS Session Key", which Bob acquired by decrypting Message B.

Step 4

The Ticket Granting Server will then decrypt the TGT in Message D using a secret key created with the Ticket Granting Server's own password (which the Authentication Server used to encrypt it), and acquire Bob's name and the "Bob-to-TGS Session Key". With the "Bob-to-TGS Session Key" it can then decrypt Message E, and if:

- The username in Message E matches the username in Message D, and
- The timestamp in Message E is within a tolerable level (usually 5 minutes)

…then the authentication process will continue forward.

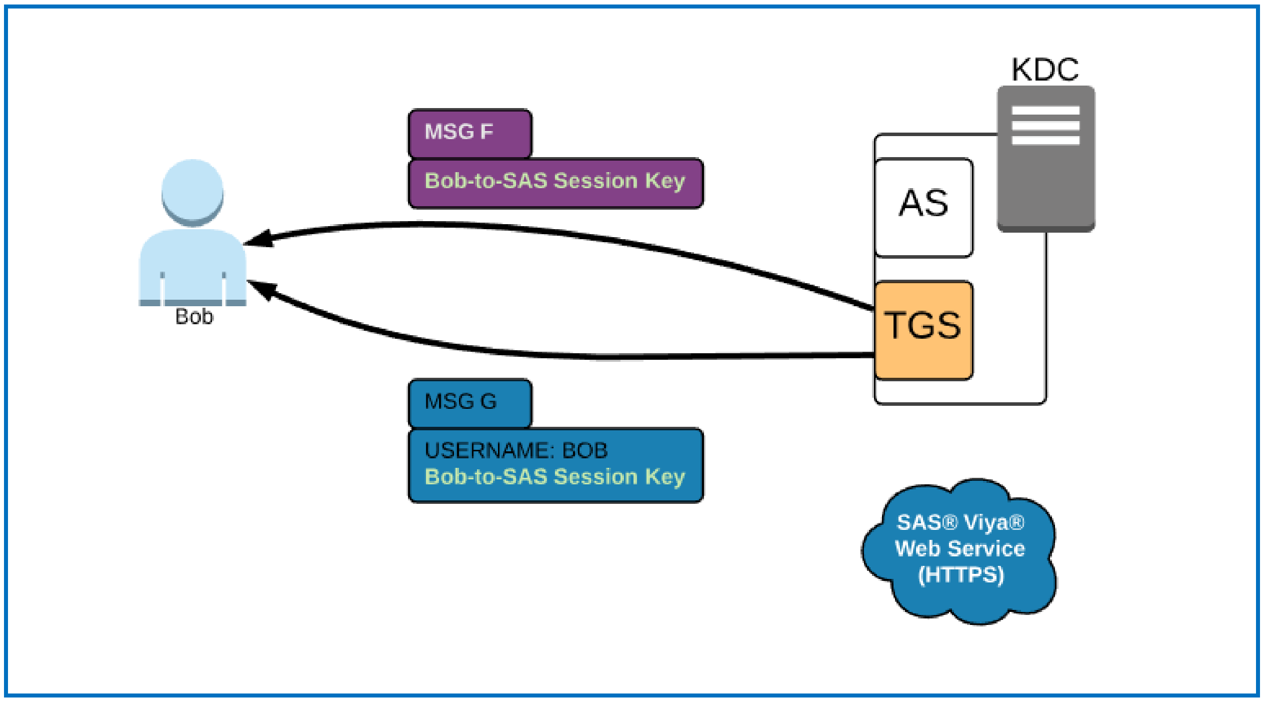


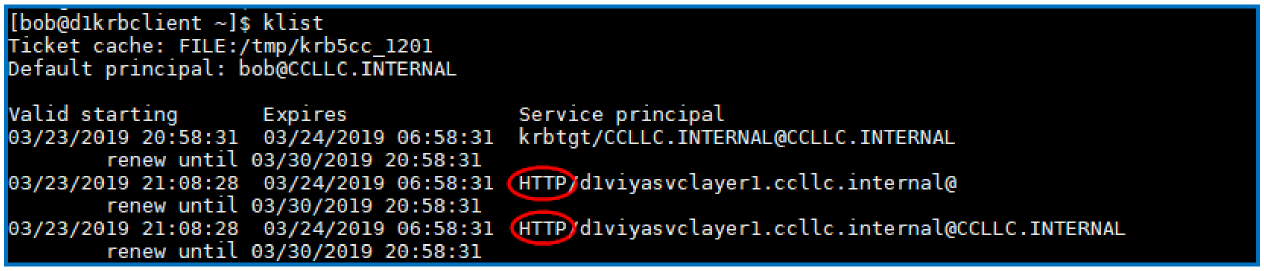
Figure 4. KDC Ticket Granting Server Reply To User (With Client-to-Server Key)

The TGS will return Bob two additional messages (see Figure 4).

Message F will be a "Bob-to-SAS Session Key" that Bob can use to talk securely to the SAS application, encrypted with the "Bob-to-TGS Session Key". Bob is able to decrypt this message immediately.

Message G is encrypted with a secret key created with the password of the SAS Application (which the KDC Ticket Granting Server knows, but Bob does not). The message contains Bob's username and a copy of the "Bob-to-SAS Session Key". This is known as the "Client-to-Server" ticket in Kerberos documentation.

Bob's klist will now look like Output 5.



```
[bob@d1krbclient ~]$ klist
Ticket cache: FILE:/tmp/krb5cc_1201
Default principal: bob@CCLLC.INTERNAL

Valid starting       Expires              Service principal
03/23/2019 20:58:31  03/24/2019 06:58:31  krbtgt/CCLLC.INTERNAL@CCLLC.INTERNAL
        renew until 03/30/2019 20:58:31
03/23/2019 21:08:28  03/24/2019 06:58:31  HTTP/d1viyasvclayer1.ccllc.internal@
        renew until 03/30/2019 20:58:31
03/23/2019 21:08:28  03/24/2019 06:58:31  HTTP/d1viyasvclayer1.ccllc.internal@CCLLC.INTERNAL
        renew until 03/30/2019 20:58:31
```

Output 5. Klist Display Showing HTTP Service Tickets

It will show that User Principal "bob" is in possession not only of the Ticket Granting Ticket acquired earlier, but also "Client-to-Server" service tickets for the SAS application. Since the SAS Visual Analytics application is web-based, the service's name begins with "http".

Step 5

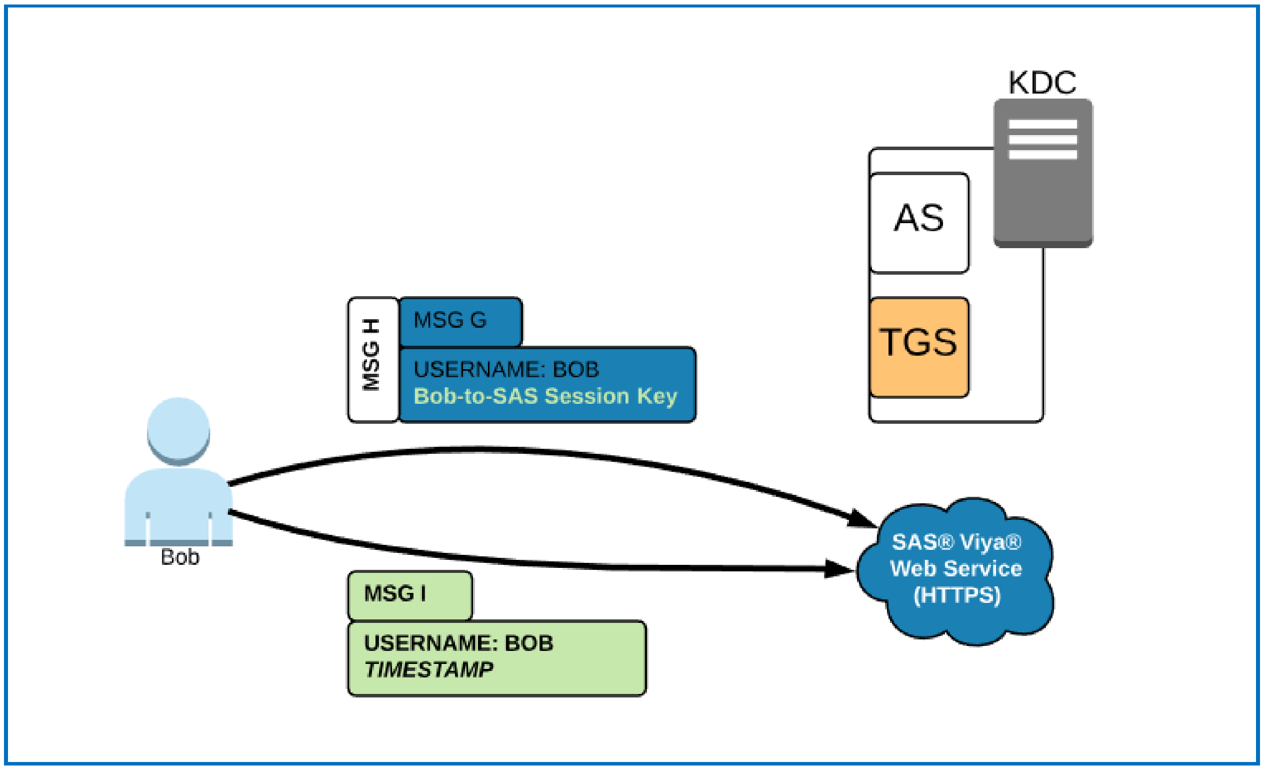


Figure 5. User Request To Kerberized Application Server For Access

Bob will then send two messages to the SAS Application (see Figure 5).

Message H is simply passing *Message G* (the Client-to-Server ticket) over to SAS, which is encrypted with the SAS Application's password (which Bob didn't know).

Message I is another "Authenticator": Bob's name and a timestamp, but this time encrypted with the "Bob-to-SAS Session Key", which Bob acquired by decrypting *Message F*.

The SAS Application Server will then decrypt the Client-to-Server ticket in *Message H*, using the secret key created with the SAS Application's own password (which the Ticket Granting Server used to encrypt the message), and acquire Bob's name and the "Bob-to-SAS Session Key". With the "Bob-to-SAS Session Key" it can then decrypt *Message I*, and if:

- The username in *Message H* matches the username in *Message I,* and
- The timestamp in *Message I* is within a tolerable level (usually 5 minutes)

…then the SAS Application will consider Bob "authenticated", and Bob will be allowed to access his SAS applications.

Success!

KEYTABS

With our Kerberos authentication example complete, we need to cover one more basic Kerberos topic. *Message B* above was encrypted with a secret key based on Bob's password, and it was decrypted by having Bob type his password in at the command line when he ran "kinit". The Ticket Granting Server is part of the KDC, and so it can always access its password to decrypt the TGT in *Message D*. But how did the SAS Application know its password, to decrypt the Client-to-Server ticket in *Message H*?

Kerberos uses a file called a "keytab" to help application services with this issue. A keytab stores pairs of principals and encrypted passwords in a non-human-readable form in a file on a filesystem, and these principal/password pairs can be used to encrypt and decrypt messages without being prompted for the password. When a kerberized service starts on Linux, it references its private keytab to acquire what it needs to encrypt/decrypt messages. We'll discuss keytabs in more detail later in the paper, but for now just be aware that kerberized services such as ssh, SAS, http, and others need a keytab file on Linux in order to perform their function in Step 5 above.

LINUX OPERATING SYSTEM CONFIGURATION

BASIC PREREQUISITES

We'll need a few prerequisites configured before we can move forward with our Kerberos-enabled host setup:

1. The fully-qualified domain name (FQDN) of each of our servers must be registered and at least forward-resolvable in DNS, as Kerberos uses DNS extensively during processing and communication. Depending on your organization, this may require a request to your "DNS Team", or it may be handled automatically during your server's registration to an AD domain (below).
2. NTP (Network Time Protocol) must be installed and running on the server, in order to ensure an accurate server clock. This is critical in order to ensure that the timestamps on the Kerberos communications we discussed above are within allowed tolerances, which Kerberos uses to protect against replay-attacks. The ntp package can be installed with yum:

```
yum install ntp
```

3. Where any installations of Java are involved during the Hadoop and SAS deployment process, if a version lower than 1.8u161 is used, the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files must be installed. These are necessary to support AES-256 encryption by Kerberos. From version 1.8u161 onward, Java includes the Policy Files in its software distribution.

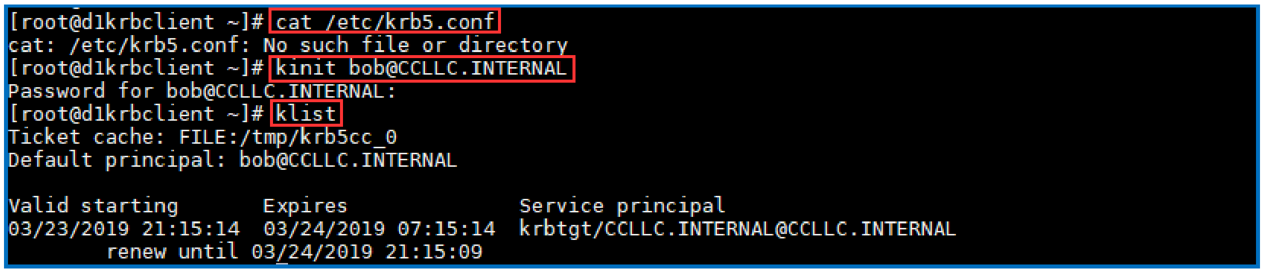
KRB5-WORKSTATION

Next, we need our server to be able to "speak" Kerberos. This functionality is provided by the krb5-workstation and krb5-libs rpm packages. Install the packages with yum:

```
yum install krb5-workstation krb5-libs
```

These two packages provide key Kerberos tools such as kinit (to obtain and store TGTs), klist (to show acquired TGTs and service tickets), ktutil (to create and manage keytabs), and kdestroy (to remove TGTs and service tickets), as well as critical shared libraries and configuration files needed to support Kerberos functionality on the server.

Note that even with a missing master Kerberos configuration file (/etc/krb5.conf, which we'll discuss shortly), the toolkit loaded by krb5-workstation and krb5-libs allows our server to communicate with a Kerberos realm. Assuming our Kerberos realm has DNS "SRV" records in place to help clients find the KDC (automatically set up with AD Kerberos, common in enterprise deployments of MIT Kerberos), we could acquire a TGT for a user (see Output 6).

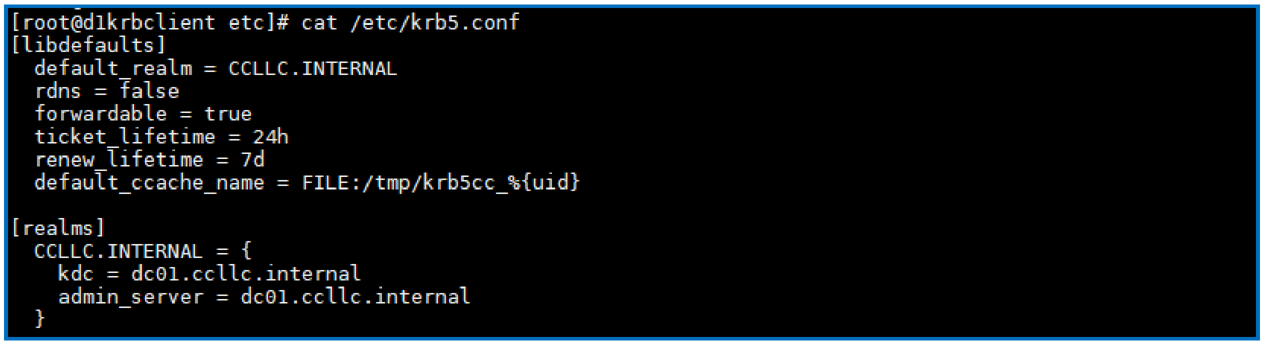


```
[root@d1krbclient ~]# cat /etc/krb5.conf
cat: /etc/krb5.conf: No such file or directory
[root@d1krbclient ~]# kinit bob@CCLLC.INTERNAL
Password for bob@CCLLC.INTERNAL:
[root@d1krbclient ~]# klist
Ticket cache: FILE:/tmp/krb5cc_0
Default principal: bob@CCLLC.INTERNAL

Valid starting       Expires              Service principal
03/23/2019 21:15:14  03/24/2019 07:15:14  krbtgt/CCLLC.INTERNAL@CCLLC.INTERNAL
        renew until 03/24/2019 21:15:09
```

Output 6. User Acquiring A TGT With A Minimal Kerberos Configuration

/ETC/KRB5.CONF

The /etc/krb5.conf file is the main Kerberos configuration file. While the file can be long and confusing in some environments, Output 7 displays a simple version which covers the requirements for our paper's SAS/Hadoop environment.



```
[root@d1krbclient etc]# cat /etc/krb5.conf
[libdefaults]
  default_realm = CCLLC.INTERNAL
  rdns = false
  forwardable = true
  ticket_lifetime = 24h
  renew_lifetime = 7d
  default_ccache_name = FILE:/tmp/krb5cc_%{uid}

[realms]
  CCLLC.INTERNAL = {
    kdc = dc01.ccllc.internal
    admin_server = dc01.ccllc.internal
  }
```

Output 7. Kerberos Master Configuration File Contents (/etc/krb5.conf)

Entries:

- **default_realm**: When a specific "@REALM" is not specified in a Kerberos command, this realm will be assumed. CCLLC.INTERNAL is the realm name of our AD domain in this paper.
- **rdns**: Enterprises generally have control of their "Forward DNS" (resolving a hostname into an IP address), but they frequently do not control their "Reverse DNS" (resolving IP addresses back to hostnames). Setting rdns=false allows Kerberos to function in such

environments, where the Forward DNS and Reverse DNS entries are not in-sync.

- **forwardable**: Allows the KDC to automatically issue a new TGT for a different host, based on an original "forwardable" TGT. See the discussion of our ssh configuration below for how this feature will be used.
- **ticket_lifetime/renew_lifetime**: Attempts to acquire TGTs with a longer active/renewable lifetime than the general (short) KDC defaults, in order to ease TGT management for users. Maximum values can be set at the KDC, however, and so there is no guarantee that the requested values will be honored.
- **default_ccache_name**: A pattern for the name of the file that will store acquired TGTs and service tickets. Although the value above is generally the default, it is good practice to explicitly set the value as we'll need to sync it with the pam_krb5/SSSD configuration later.
- **[realms] section**: Establishes various properties for each Kerberos realm, such as where the Realm KDC and Admin Server are located on the network.

JOINING A REALM/DOMAIN

The simple configuration above could be leveraged to provide access to a remote kerberized Hadoop instance from an unkerberized SAS environment, because we are able to acquire a valid TGT which could be used downstream to authenticate to the Hadoop instance.

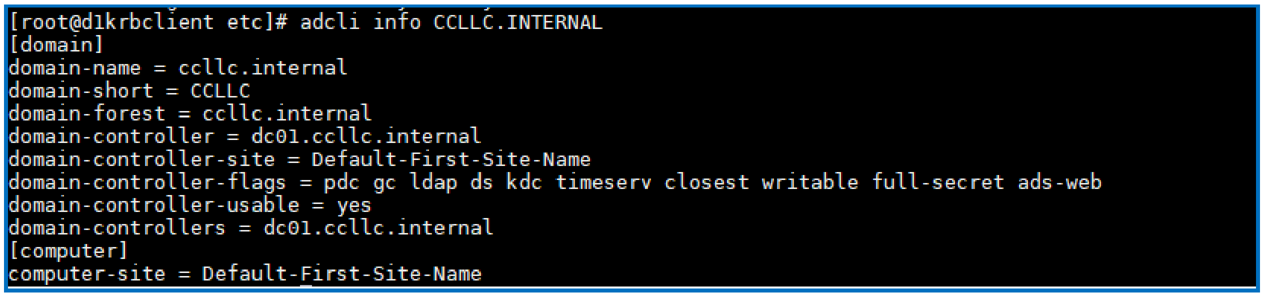
In this paper's environment, however, we will have a fully-kerberized SAS and Hadoop stack local on the servers. In order to support this, we will want to "register" the server in the Kerberos realm, known as "joining the domain" to Active Directory, or "adding the host" to an MIT realm. Once the server is added to the Kerberos realm, it can serve as a platform for hosting kerberized OS-layer services that we'll need later, such as local passwordless ssh logins using Kerberos tickets.

The easiest way to join an AD domain on CentOS or RedHat is to use the "realmd" rpm package, available via yum. In one step, it can join the domain as well as implement the SSSD configuration we'll cover later. However, using realmd as an "easy button" for our paper will make it more difficult to truly understand how our Kerberos configuration works, and so we'll instead use the individual commands that realmd uses under the covers.

We start with the "adcli" (AD Command Line Interface) tool to join the domain, available via yum:

```
yum install adcli
```

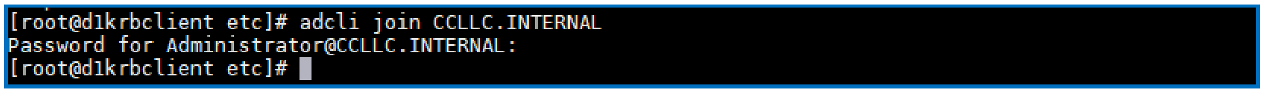
The adcli command can acquire some basic information about our "ccllc.internal" AD domain before we even join (see Output 8).



```
[root@d1krbclient etc]# adcli info CCLLC.INTERNAL
[domain]
domain-name = ccllc.internal
domain-short = CCLLC
domain-forest = ccllc.internal
domain-controller = dc01.ccllc.internal
domain-controller-site = Default-First-Site-Name
domain-controller-flags = pdc gc ldap ds kdc timeserv closest writable full-secret ads-web
domain-controller-usable = yes
domain-controllers = dc01.ccllc.internal
[computer]
computer-site = Default-First-Site-Name
```

Output 8. Using adcli To Acquire Information About An AD Domain

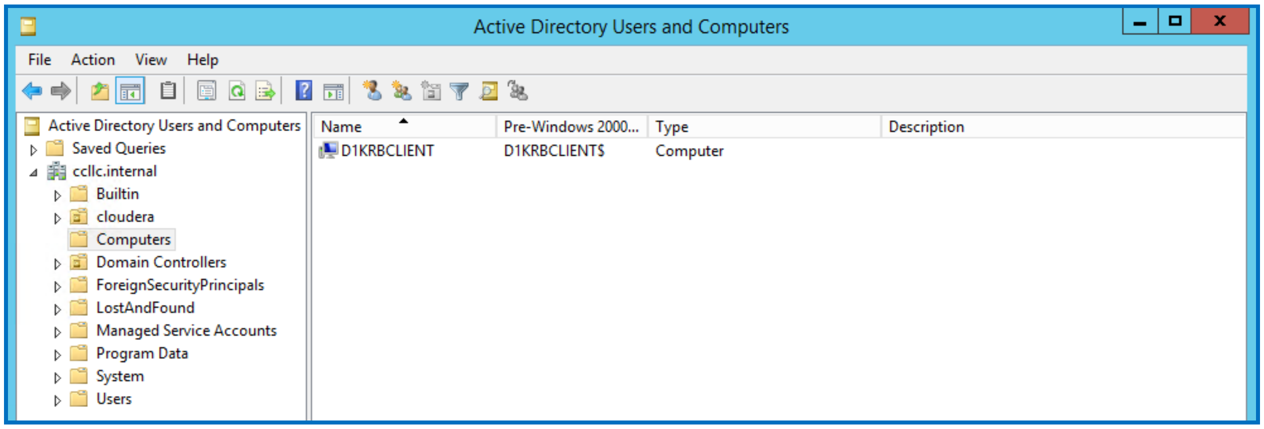
Then it can be used to register our host in AD, using AD's Administrator password (see Output 9).



```
[root@d1krbclient etc]# adcli join CCLLC.INTERNAL
Password for Administrator@CCLLC.INTERNAL:
[root@d1krbclient etc]# 
```

Output 9. Using adcli To Join An AD Domain

After registration, our host will appear in our AD domain (see Display 1).



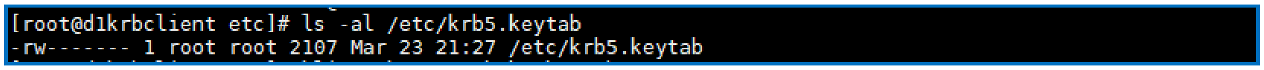
```
Active Directory Users and Computers
  Saved Queries
  ccllc.internal
    Builtin
    cloudera
    Computers
    Domain Controllers
    ForeignSecurityPrincipals
    LostAndFound
    Managed Service Accounts
    Program Data
    System
    Users
```

Name	Pre-Windows 2000...	Type	Description
D1KRBCLIENT	D1KRBCLIENT$	Computer	

Display 1. A Host After Joining An AD Domain

HOST KEYTAB

Joining the domain above automatically places a keytab file at /etc/krb5.keytab on a Linux host and protects it with root-only access (see Output 10).

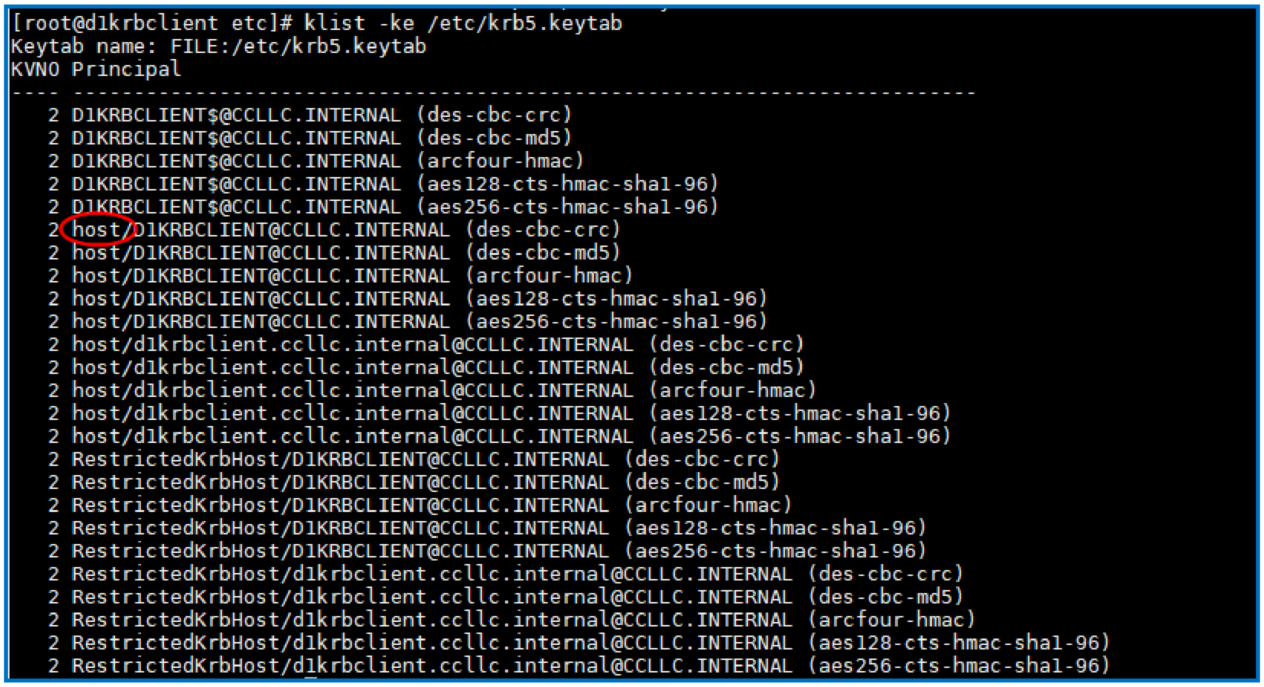


```
[root@d1krbclient etc]# ls -al /etc/krb5.keytab
-rw------- 1 root root 2107 Mar 23 21:27 /etc/krb5.keytab
```

Output 10. Host Keytab Location And Permissions

As discussed above, a keytab is used by services to be able to decrypt messages created by the KDC, shown in Step 5 of our Kerberos Authentication Example above. This particular keytab is called the "Host Keytab", and it is used to validate kerberized access to specific OS-layer services such as ssh that run as root on our Linux server.

A keytab's contents can be examined using the klist command. In Output 11 we can see that Kerberos Services with primary name "host" and an instance name referring to host d1krbclient in our environment are among those which can be decrypted and validated with our Host Keytab.

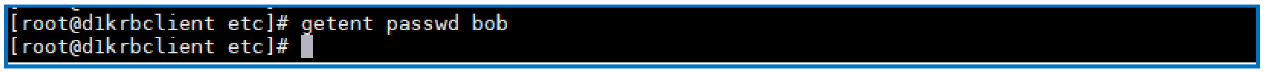


```
[root@d1krbclient etc]# klist -ke /etc/krb5.keytab
Keytab name: FILE:/etc/krb5.keytab
KVNO Principal
---- --------------------------------------------------------------------------
   2 D1KRBCLIENT$@CCLLC.INTERNAL (des-cbc-crc)
   2 D1KRBCLIENT$@CCLLC.INTERNAL (des-cbc-md5)
   2 D1KRBCLIENT$@CCLLC.INTERNAL (arcfour-hmac)
   2 D1KRBCLIENT$@CCLLC.INTERNAL (aes128-cts-hmac-sha1-96)
   2 D1KRBCLIENT$@CCLLC.INTERNAL (aes256-cts-hmac-sha1-96)
   2 host/D1KRBCLIENT@CCLLC.INTERNAL (des-cbc-crc)
   2 host/D1KRBCLIENT@CCLLC.INTERNAL (des-cbc-md5)
   2 host/D1KRBCLIENT@CCLLC.INTERNAL (arcfour-hmac)
   2 host/D1KRBCLIENT@CCLLC.INTERNAL (aes128-cts-hmac-sha1-96)
   2 host/D1KRBCLIENT@CCLLC.INTERNAL (aes256-cts-hmac-sha1-96)
   2 host/d1krbclient.ccllc.internal@CCLLC.INTERNAL (des-cbc-crc)
   2 host/d1krbclient.ccllc.internal@CCLLC.INTERNAL (des-cbc-md5)
   2 host/d1krbclient.ccllc.internal@CCLLC.INTERNAL (arcfour-hmac)
   2 host/d1krbclient.ccllc.internal@CCLLC.INTERNAL (aes128-cts-hmac-sha1-96)
   2 host/d1krbclient.ccllc.internal@CCLLC.INTERNAL (aes256-cts-hmac-sha1-96)
   2 RestrictedKrbHost/D1KRBCLIENT@CCLLC.INTERNAL (des-cbc-crc)
   2 RestrictedKrbHost/D1KRBCLIENT@CCLLC.INTERNAL (des-cbc-md5)
   2 RestrictedKrbHost/D1KRBCLIENT@CCLLC.INTERNAL (arcfour-hmac)
   2 RestrictedKrbHost/D1KRBCLIENT@CCLLC.INTERNAL (aes128-cts-hmac-sha1-96)
   2 RestrictedKrbHost/D1KRBCLIENT@CCLLC.INTERNAL (aes256-cts-hmac-sha1-96)
   2 RestrictedKrbHost/d1krbclient.ccllc.internal@CCLLC.INTERNAL (des-cbc-crc)
   2 RestrictedKrbHost/d1krbclient.ccllc.internal@CCLLC.INTERNAL (des-cbc-md5)
   2 RestrictedKrbHost/d1krbclient.ccllc.internal@CCLLC.INTERNAL (arcfour-hmac)
   2 RestrictedKrbHost/d1krbclient.ccllc.internal@CCLLC.INTERNAL (aes128-cts-hmac-sha1-96)
   2 RestrictedKrbHost/d1krbclient.ccllc.internal@CCLLC.INTERNAL (aes256-cts-hmac-sha1-96)
```

Output 11. Host Keytab Contents

IDENTIFICATION AND AUTHORIZATION

Even though we've added our computer to the AD domain, and we have a host keytab that could be used by ssh to validate Kerberos tickets, our server still can't "see" the users and groups in the AD domain as needed for them to actually log in (see Output 12).



```
[root@d1krbclient etc]# getent passwd bob
[root@d1krbclient etc]# 
```

Output 12. Failed Check For A User's Existence At The Host Level

In order to enable logins with AD usernames and passwords, we still need to configure our server to *identify* the members of the AD domain and their Unix attributes (uid, gid, home directory, shell), and to *authenticate* them using Kerberos.

There are a number of systems available which can contain users and their attributes for identification on a Linux host, but LDAP is by far the most common central repository for enterprise environments. LDAP also happens to be a hard-requirement for full deployments of Viya software, and therefore we will be using it for user identification in our implementation. Note that there are a number of different LDAP variants on the market, including OpenLDAP, Apache Directory Server, Oracle Directory Server, and Microsoft Active Directory.

There are also a number of Kerberos variants on the market, including MIT Kerberos, Heimdal Kerberos, and the Kerberos that comes built-in with Microsoft Active Directory. As Active Directory is able to provide both LDAP and Kerberos services to our environment and is a very popular deployment choice in enterprise environments, we will be using it in this paper. Your company may use a different mix (e.g., OpenLDAP and MIT Kerberos), but the principles of the implementation will remain the same.

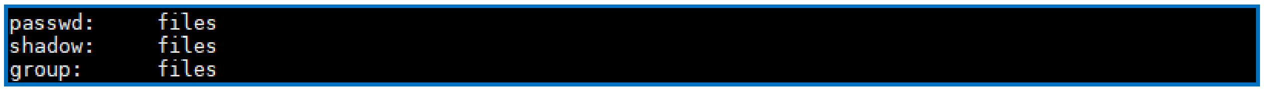
NSS AND PAM

While applications can use their own custom methods for identification and authentication, software running on Unix/Linux frequently takes advantage of a pair of subsystems that provide a common toolkit for identification and authentication. These are:

- The Name Service Switch (NSS) subsystem which provides a common set of APIs for identifying users and groups, and is primarily configured with the nsswitch.conf file.
- The Pluggable Authentication Module (PAM) subsystem which provides a standard framework for user authentication, and is configured with various files in the /etc/pam.d directory.

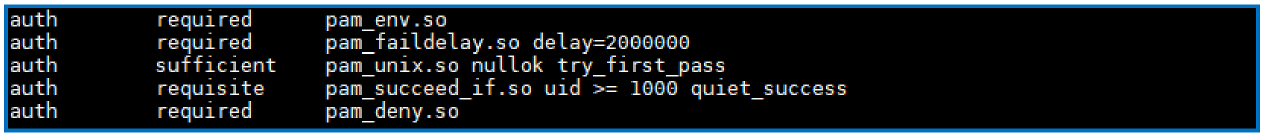
During a standard Viya install, one of the first tasks after the software is installed is to configure a direct connection to an LDAP provider in Environment Manager, which bypasses NSS and PAM for identification and authentication into the Viya visual interfaces. However, many of the popular SAS Viya-based product offerings require certain operating system processes to be launched as the ID of the requesting SAS user, such SAS Studio 4.x and 5.x, SAS Visual Data Mining and Machine Learning, SAS Model Manager, SAS Visual Forecasting, and SAS Decision Manager. NSS and PAM need to be configured properly in order to support this functionality, and adding Kerberos authentication brings specific requirements to the NSS/PAM configurations. Note that we'll also cover how to configure Kerberos authentication into the visual interfaces (via IWA) later in the paper.

To show an example configuration of NSS and PAM, Output 13 and Output 14 show selected contents of the key NSS and PAM configuration files on a CentOS system with no enterprise integration at all.



```
passwd:     files
shadow:     files
group:      files
```

Output 13. Selected Contents Of The /etc/nsswitch.conf File After OS Installation



```
auth        required      pam_env.so
auth        required      pam_faildelay.so delay=2000000
auth        sufficient    pam_unix.so nullok try_first_pass
auth        requisite     pam_succeed_if.so uid >= 1000 quiet_success
auth        required      pam_deny.so
```

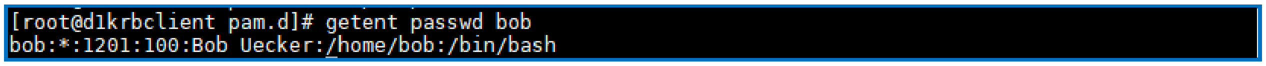
Output 14. Selected Contents Of The /etc/pam.d/password-auth File After OS Installation

- The nsswitch.conf file, which controls the user identification process, refers all identification requests to "files". This means that it looks only to the local /etc/passwd, /etc/shadow, and /etc/group files for information.
- While understanding PAM files can be its own art, the key line above is the "pam_unix.so" entry. The pam_unix library is the standard Unix authentication module, and it looks as well to local files such as /etc/passwd and /etc/shadow for authentication information.

Combining our new understanding of the roles of NSS and PAM with our need for LDAP identification and Kerberos authentication, we can state the following:

- We need to configure the nsswitch.conf so that NSS can talk to an LDAP, either directly or via a system that communicates with LDAP on NSS's behalf.
- We need to configure PAM so that it can talk to a Kerberos realm, either directly or via a system that communicates with a Kerberos realm on PAM's behalf.

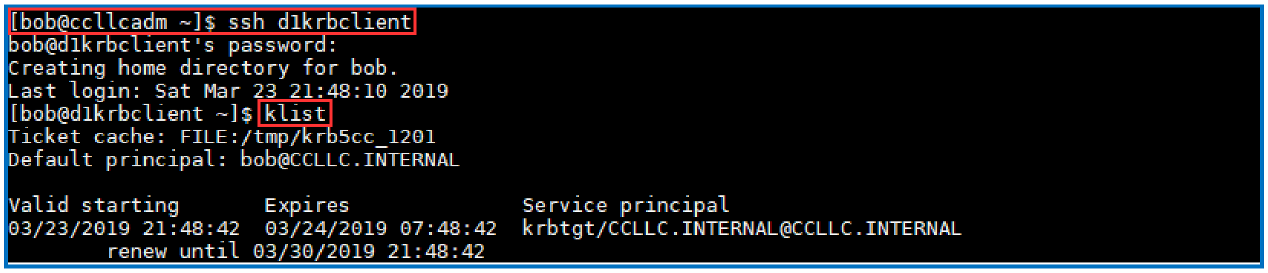
We outline three different options for configuring NSS and PAM below. At the end of each, we should be able to "see" bob as an available ID on the system (see Output 15).



```
[root@d1krbclient pam.d]# getent passwd bob
bob:*:1201:100:Bob Uecker:/home/bob:/bin/bash
```

Output 15. Successful Check For A User's Existence At The Host Level

Bob should also be able to log in to the server using his AD username and password, have a home directory created if he needs one, and have a Kerberos TGT created automatically which he can use to gain access to kerberized applications such as Hadoop in his enterprise environment (see Output 16).



```
[bob@ccllcadm ~]$ ssh d1krbclient
bob@d1krbclient's password:
Creating home directory for bob.
Last login: Sat Mar 23 21:48:10 2019
[bob@d1krbclient ~]$ klist
Ticket cache: FILE:/tmp/krb5cc_1201
Default principal: bob@CCLLC.INTERNAL

Valid starting       Expires              Service principal
03/23/2019 21:48:42  03/24/2019 07:48:42  krbtgt/CCLLC.INTERNAL@CCLLC.INTERNAL
        renew until 03/30/2019 21:48:42
```

Output 16. User Accessing A Server Via SSH With NSS/PAM Configured For AD Authentication

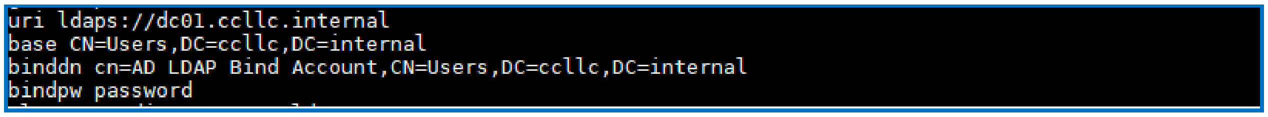
NSS/PAM OPTION 1: NSS_LDAP AND PAM_KRB5

In order to have direct connections to LDAP and Kerberos from NSS and PAM, we'll load a specific pair of NSS/PAM rpm packages, along with a companion "oddjob-mkhomedir" package:

```
yum install nss-pam-ldapd pam_krb5 oddjob-mkhomedir
```

The oddjob-mkhomedir package will be used to automatically create a home directory for our users upon first access, if they do not already have one.

With our krb5.conf file already in place for Kerberos, if we have a valid CA certificate for our KDC's TLS-secured LDAP configured on our host in /etc/openldap/certs, we can simply uncomment the "Mappings for Active Directory" section of the default /etc/nslcd.conf file (which configures the local LDAP Name Service Daemon: nslcd) and update just a few other lines to tell the daemon how to connect to our AD LDAP (see Output 17).



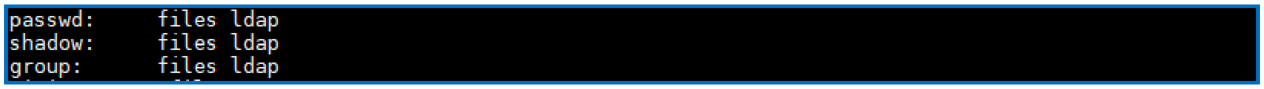
```
uri ldaps://dc01.ccllc.internal
base CN=Users,DC=ccllc,DC=internal
binddn cn=AD LDAP Bind Account,CN=Users,DC=ccllc,DC=internal
bindpw password
```

Output 17. Selected Contents Of /etc/nslcd.conf For Access To AD LDAP

Then we can run a few commands to enable our nss_ldap/pam_krb5 configuration:

```
systemctl enable nslcd

systemctl start nslcd

authconfig --enableldap --enablekrb5 --enablemkhomedir --update
```

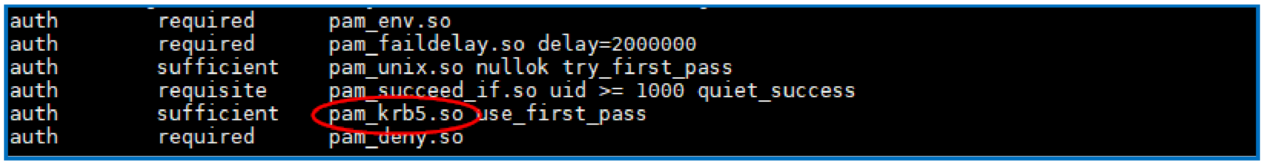
After running these commands, our nsswitch.conf will now include "ldap" as a target for passwd/shadow/group queries (identification), as shown in Output 18.



```
passwd:     files ldap
shadow:     files ldap
group:      files ldap
```

Output 18. Selected Contents Of The /etc/nsswitch.conf File After Authconfig Enableldap/krb5

Additionally, a number of files in the pam.d directory will be updated to support Kerberos authentication with pam_krb5. Looking at just the top of the /etc/pam.d/password_auth file in Output 19, for example, we can now see that the "pam_krb5.so" module has been added to our system's authorization "stack".



```
auth        required      pam_env.so
auth        required      pam_faildelay.so delay=2000000
auth        sufficient    pam_unix.so nullok try_first_pass
auth        requisite     pam_succeed_if.so uid >= 1000 quiet_success
auth        sufficient    pam_krb5.so use_first_pass
auth        required      pam_deny.so
```

Output 19. Selected Contents Of The password_auth PAM File After Authconfig Enableldap/krb5

NSS/PAM OPTION 2: SSSD

While using nss_ldap and pam_krb5 remains a common configuration to implement LDAP identification and Kerberos authentication in enterprise environments, RedHat deprecated pam_krb5 with the RedHat Enterprise Linux 7.4 (RHEL) release and does not plan to include pam_krb5 with RHEL 8. The target replacement for pam_krb5, which is already available in both RHEL 6 and RHEL 7, is the System Security Service Daemon (SSSD). SSSD implements features which are not available in a standard nss_ldap/pam_krb5 configuration, such as authentication when AD is offline, better performance and LDAP ID-mapping (discussed later).

Its architecture is to reside between NSS/PAM and the LDAP/Kerberos systems, and to make requests to LDAP/Kerberos on behalf of NSS/PAM as shown in Figure 6.

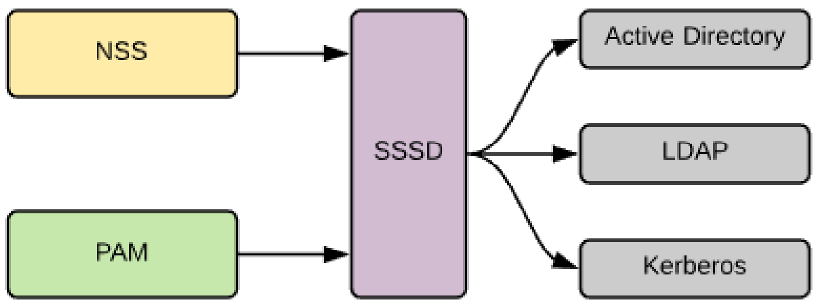


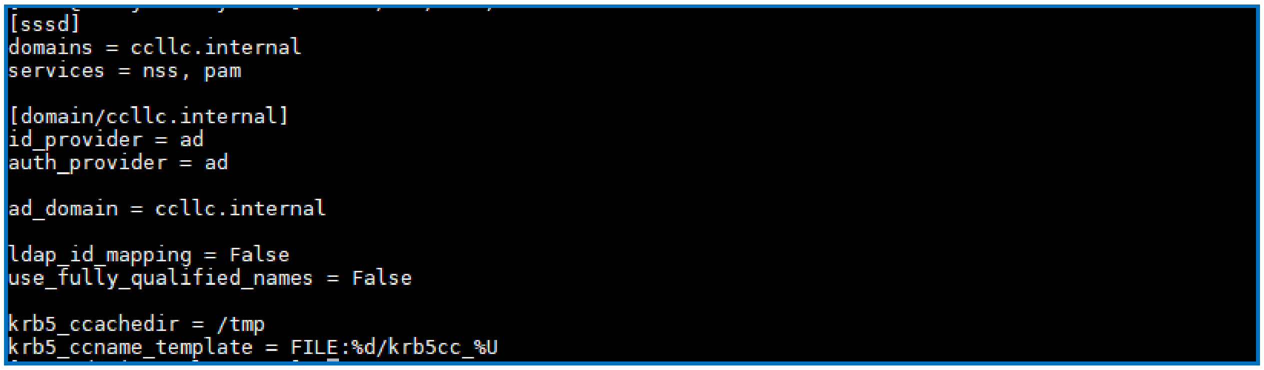
Figure 6. SSSD Integration Architecture With NSS/PAM and AD

For configuration of SSSD in non-AD Kerberos environments, the "sssd-ldap" and "sssd-krb5" rpm packages can be installed with yum and then configured. However, SSSD offers a dedicated "sssd-ad" rpm package for Active Directory environments, which includes specific optimizations for communicating with AD infrastructure. We'll install sssd-ad for our paper's environment, along with a companion "oddjob-mkhomedir" package:

```
yum install sssd-ad oddjob-mkhomedir
```

The oddjob-mkhomedir package will be used to automatically create a home directory for our users upon first access, if they do not already have one.

Like the krb5.conf, sssd's main configuration file at /etc/sssd/sssd.conf can be long and confusing in some environments. However, only a few core lines are actually needed in our paper's environment, as shown in Output 20.



```
[sssd]
domains = ccllc.internal
services = nss, pam

[domain/ccllc.internal]
id_provider = ad
auth_provider = ad

ad_domain = ccllc.internal

ldap_id_mapping = False
use_fully_qualified_names = False

krb5_ccachedir = /tmp
krb5_ccname_template = FILE:%d/krb5cc_%U
```

Output 20. SSSD Master Configuration File Contents (/etc/sssd/sssd.conf)

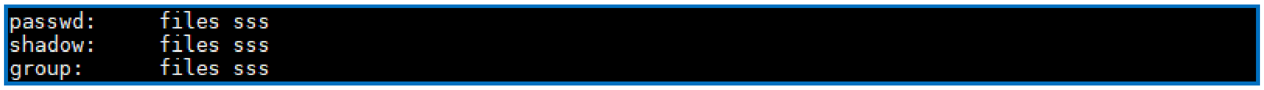
Entries:

- **[sssd] section:** Configures ccllc.internal as one available domain to use for user identification and authorization (sssd can support multiple user repositories), and also sets up sssd to service both nss and pam system calls.
- **[domain/ccllc.internal] section:** Configuration specific to the ccllc.internal domain.
- **id_provider/auth_provider:** Sets SSSD to use our AD domain for both identification and authorization purposes.
- **ldap_id_mapping:** Supports automatic translation of Active Directory User/Group SIDs into Unix UIDs/GIDs. Kerberos will function properly with this setting true or false. In our paper's environment, we manually managed uids/gids/shells from each user's "Attribute Editor" tab in the Windows "Active Directory Users and Computers" application, so that the uids/gids would match in both the nss_ldap/pam_krb5 and the SSSD configurations (nss_ldap does not support LDAP ID mapping). But most enterprise environments with Active Directory will have ldap_id_mapping set to true.
- **krb5_ccachedir/krb5_ccname_template:** These settings sync the location where Kerberos TGT and service tickets will be stored, with the krb5.conf file which we configured earlier. Having the two files in-sync is important for SAS, so that when a Kerberos TGT is acquired by authenticating through PAM/SSSD and stored on the filesystem, SAS applications (who look in the default_ccache_name location we defined in the krb5.conf file) are able to "find" the TGT.

With our /etc/sssd/sssd.conf file in-place, we can now enable SSSD to service NSS identification requests, Kerberos authorization requests, and create home directories for new users with the following:

```
systemctl enable sssd

systemctl start sssd

authconfig --enablesssd --enablesssdauth --enablemkhomedir –update
```

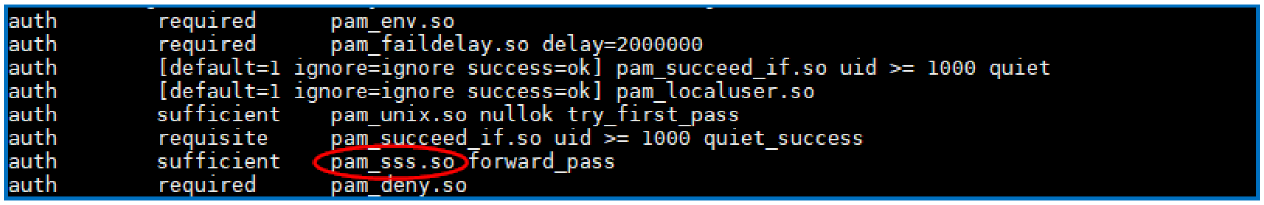
After running these commands, our nsswitch.conf will now include "sss" as a target for passwd/shadow/group queries (identification), as shown in Output 21.



```
passwd:     files sss
shadow:     files sss
group:      files sss
```

Output 21. Selected Contents Of The /etc/nsswitch.conf File After Authconfig Enablesssd

Also, a number of files in the pam.d directory will be updated to support Kerberos authentication managed by SSSD. Looking at just the top of the /etc/pam.d/password_auth file in Output 22, for example, we can now see that the "pam_sss.so" module has been added to our system's authorization "stack".



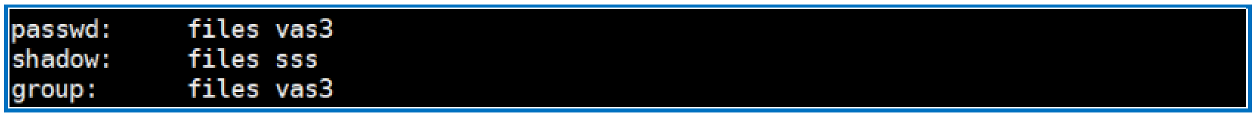
```
auth        required      pam_env.so
auth        required      pam_faildelay.so delay=2000000
auth        [default=1 ignore=ignore success=ok] pam_succeed_if.so uid >= 1000 quiet
auth        [default=1 ignore=ignore success=ok] pam_localuser.so
auth        sufficient    pam_unix.so nullok try_first_pass
auth        requisite     pam_succeed_if.so uid >= 1000 quiet_success
auth        sufficient    pam_sss.so forward_pass
auth        required      pam_deny.so
```

Output 22. Selected Contents Of The password_auth PAM File After Authconfig Enablesssd

NSS/PAM OPTION 3: VAS, WINBIND, CENTRIFY

There are a number of other products on the market for integrating Linux systems with Active Directory domains, including VAS, Winbind, and Centrify. These all work in a similar fashion to SSSD, in that NSS and PAM make calls to the product when they need identification or authorization support, and the product contacts AD on NSS/PAM's behalf. Many offer similar features to SSSD, such as authentication when AD is offline, better performance and LDAP ID-mapping.

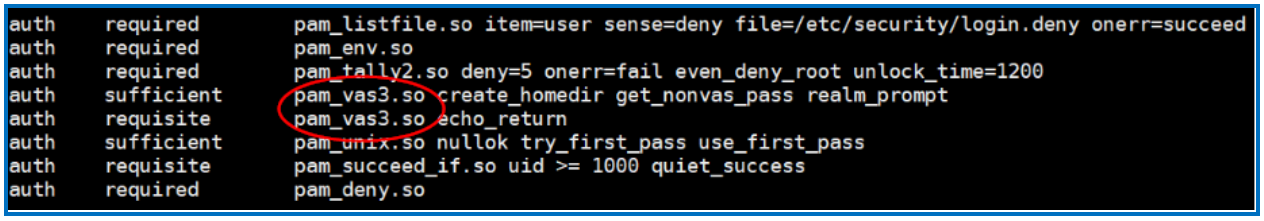
The principles remain the same as SSSD for configuration as well. For example, VAS uses its own command called "vastool" instead of "authconfig" to enable VAS, but after running the vastool command there will be updates to /etc/nsswitch.conf for user identification as in Output 23.



```
passwd:     files vas3
shadow:     files sss
group:      files vas3
```

Output 23. Selected Contents Of The /etc/nsswitch.conf File After Vastool Execution

There will also be updates to various files in /etc/pam.d for user authorization, for example the update to /etc/pam.d/password_auth shown in Output 24.



```
auth    required        pam_listfile.so item=user sense=deny file=/etc/security/login.deny onerr=succeed
auth    required        pam_env.so
auth    required        pam_tally2.so deny=5 onerr=fail even_deny_root unlock_time=1200
auth    sufficient      pam_vas3.so create_homedir get_nonvas_pass realm_prompt
auth    requisite       pam_vas3.so echo_return
auth    sufficient      pam_unix.so nullok try_first_pass use_first_pass
auth    requisite       pam_succeed_if.so uid >= 1000 quiet_success
auth    required        pam_deny.so
```

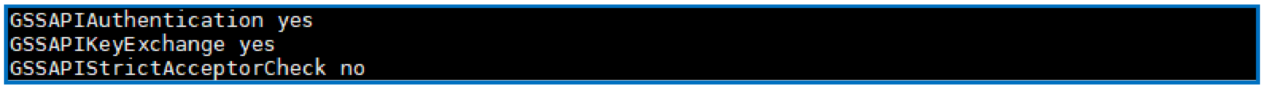
Output 24. Selected Contents Of The password_auth PAM File After Vastool Execution

SSH KERBEROS CONFIGURATION

At this point, Bob is able to log into our server successfully using his username and password, and he has a Kerberos TGT created for him upon initial login. However, we are not able to use that Kerberos TGT to log in passwordless to the server itself, and we'll need to enable that functionality in order to access and use our kerberized Hadoop cluster. To do that, we'll change a group of "GSSAPI" settings in the /etc/sshd/sshd_config and /etc/sshd/ssh_config files.

What is GSSAPI? GSSAPI can be used as a common programming interface for applications to make calls to Kerberos, and ensures the same application code can be used no matter which flavor or revision of Kerberos is actually being used in an environment. A SAS Administrator does not normally need to worry about the specifics of GSSAPI, and it's generally easier to figure out what a particular GSSAPI setting does by imagining the word "Kerberos" swapped for the word "GSSAPI", e.g. when you see "GSSAPIAuthentication" in a configuration file, think "this really means KerberosAuthentication".

We'll update our /etc/ssh/sshd_config file (which controls the *ssh server* for *inbound* connections) with the settings outlined in Output 25.

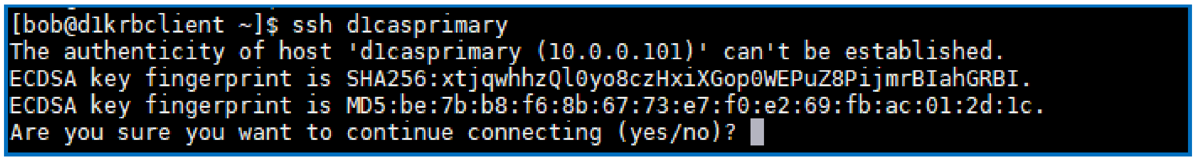


```
GSSAPIAuthentication yes
GSSAPIKeyExchange yes
GSSAPIStrictAcceptorCheck no
```

Output 25. Selected Contents Of The /etc/ssh/sshd_config File Enabling Kerberos Support

Entries:

- **<u>GSSAPIAuthentication yes:</u>** Allows the server to accept Kerberos credentials for ssh logins
- **<u>GSSAPIKeyExchange yes:</u>** Uses Kerberos to validate hosts instead of ssh host-keys, which means users will not need to worry about host-authentication prompts like the one shown in Output 26 each time they log into a server for the first time.



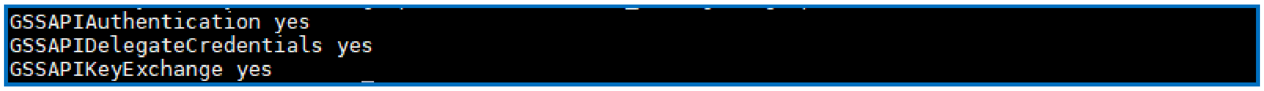
```
[bob@d1krbclient ~]$ ssh d1casprimary
The authenticity of host 'd1casprimary (10.0.0.101)' can't be established.
ECDSA key fingerprint is SHA256:xtjqwhhzQl0yo8czHxiXGop0WEPuZ8PijmrBIahGRBI.
ECDSA key fingerprint is MD5:be:7b:b8:f6:8b:67:73:e7:f0:e2:69:fb:ac:01:2d:1c.
Are you sure you want to continue connecting (yes/no)?
```

Output 26. An SSH Host Authenticity Check

- **<u>GSSAPIStrictAcceptorCheck no:</u>** In enterprise environments, hosts often have aliases in addition to their real hostnames (e.g., server "g7236039.example.org" might have an easy-to-remember alias "sascasprimary.appl.example.org" registered in DNS).

Setting the StrictAcceptorCheck to “no” allows users to ssh to alias names instead of just the real hostname, and still successfully use Kerberos tickets to authenticate.

We’ll also update the /etc/ssh/ssh_config file (which controls the *ssh client* on *outbound* connections) with the settings shown in Output 27.



```
GSSAPIAuthentication yes
GSSAPIDelegateCredentials yes
GSSAPIKeyExchange yes
```

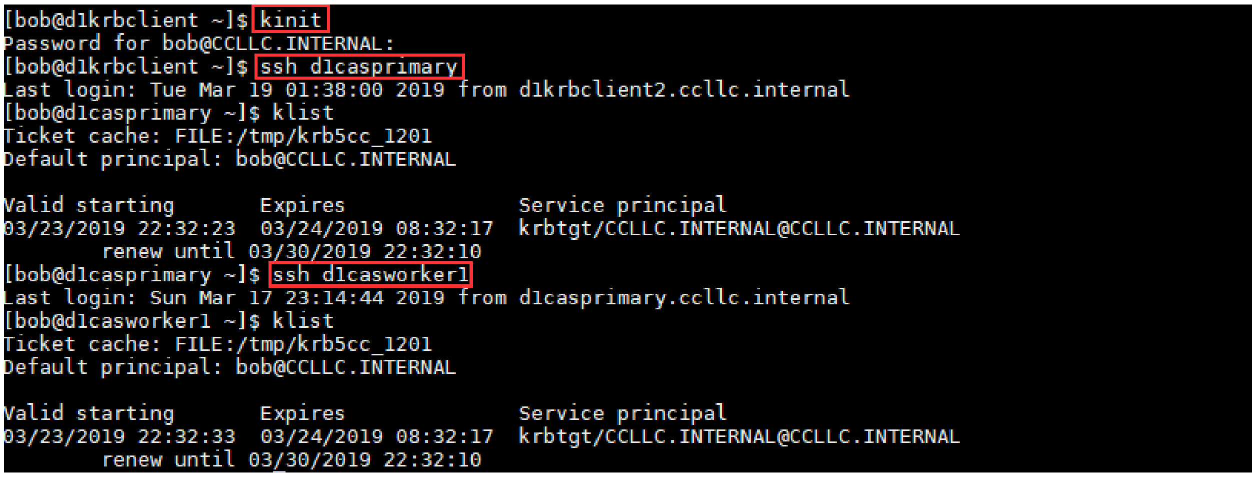
Output 27. Selected Contents Of The /etc/ssh/ssh_config File Enabling Kerberos Support

Two of the entries are the same as in our sshd_config file. The new entry is:

- **<u>GSSAPIDelegateCredentials yes:</u>** Configures the ssh client to forward Kerberos credentials to the target server, if possible.

What does forwarding credentials mean, and why do we need it? Forwarding with ssh means that after you acquire a TGT on one server, that when you log in to another server a TGT will be automatically created for you there.

Notice in the example in Output 28 that Bob acquired his original TGT on host d1krbclient, that he then ssh’ed (passwordless) to host d1casprimary where a TGT was automatically created for him, which he then used to ssh (also passwordless) into host d1casworker1, where another TGT was automatically created for him. He only needed to kinit once at the beginning of the process.



```
[bob@d1krbclient ~]$ kinit
Password for bob@CCLLC.INTERNAL:
[bob@d1krbclient ~]$ ssh d1casprimary
Last login: Tue Mar 19 01:38:00 2019 from d1krbclient2.ccllc.internal
[bob@d1casprimary ~]$ klist
Ticket cache: FILE:/tmp/krb5cc_1201
Default principal: bob@CCLLC.INTERNAL

Valid starting       Expires              Service principal
03/23/2019 22:32:23  03/24/2019 08:32:17  krbtgt/CCLLC.INTERNAL@CCLLC.INTERNAL
        renew until 03/30/2019 22:32:10
[bob@d1casprimary ~]$ ssh d1casworker1
Last login: Sun Mar 17 23:14:44 2019 from d1casprimary.ccllc.internal
[bob@d1casworker1 ~]$ klist
Ticket cache: FILE:/tmp/krb5cc_1201
Default principal: bob@CCLLC.INTERNAL

Valid starting       Expires              Service principal
03/23/2019 22:32:33  03/24/2019 08:32:17  krbtgt/CCLLC.INTERNAL@CCLLC.INTERNAL
        renew until 03/30/2019 22:32:10
```

Output 28. Multi-Server Passwordless SSH Access With Forwarded Kerberos Credentials

In order to fully implement forwarding we also included “forwardable = true” in our krb5.conf above, as “forwardable” is a TGT attribute that has to be specifically requested from the KDC.

This forwarding capability is also needed in order to support Kerberos TGT delegation for SAS and Hadoop, and will be discussed in greater detail later in the paper.

KERBEROS PRINCIPALS - DEEP DIVE

A few notes about Kerberos Principals before we proceed with our kerberization of Hadoop and SAS. The SAS Administrator will not normally need to remember this information in-depth during day-to-day support of an environment, but it is used when an environment is initially configured, and therefore it will help clarify some of the configurations we'll discuss later in the paper.

- Service Principals such as sascas/d1casprimary.ccllc.internal@CCLLC.INTERNAL (CAS on host d1casprimary) and host/d1casworker1.ccllc.internal@CCLLC.INTERNAL (OS services on d1casworker1) are "Standalone Kerberos Principals" in MIT Kerberos. All principals are the same as one another in MIT, whether they represent a user, a host, or a service on the network. All principals are also able to acquire TGTs, provided the password is known or a valid keytab is available.
- In Active Directory, however, Service Principals are an *attribute* of either a User Principal (e.g. bob@CCLLC.INTERNAL) or a Machine Principal (e.g. D1CASPRIMARY$@CCLLC.INTERNAL). Only User and Machine Principals can acquire TGTs. Service Principals cannot.
- "Machine Principal" is not the commonly used term in Active Directory documentation. A host in AD is normally referred to as a "Computer Object" or a "Machine Account". However it is important to understand that computers in an Active Directory domain are Kerberos Principals, and that they are very "User-Like" in that they can have Service Principals as attributes, and can acquire TGTs.
- In Active Directory Kerberos, all User and Machine Principal Names must be unique, and all Service Principal Names must be unique. However, a User or Machine Principal can be the same name as a Service Principal. Machine Principals traditionally end in a "$", which helps reduce naming conflicts with User Principals. In MIT Kerberos, where all User/Host/Service Principals are equal, all Principal names must be unique across all types.
- In Active Directory, a User or Machine Principal is allowed to have more than one Service Principal as an attribute. However, a Service Principal cannot be an attribute to multiple Users or Machines, as this breaks the uniqueness requirement above.
- On *Windows* servers in Active Directory environments, Service Principals associated with *User Principals* represent services that run as that User Principal account on a Windows server.
- On *Linux* servers in Active Directory environments, Service Principals associated with *User Principals* do not need to run as the User Principal account. As long as the Linux service is in possession of a valid keytab for the User Principal, the service can run under any account on the Linux host.
- On *Windows* servers in Active Directory environments, Service Principals associated with *Machine Principals* represent services that run as the LocalSystem account on the Windows server.
- On *Linux* servers in Active Directory environments, Service Principals associated with *Machine Principals* represent OS services that run as root on the Linux server and validate using the host/<fqdn>@REALM Service Principal (e.g. host/d1casworker1.ccllc.internal@CCLLC.INTERNAL). These services use the /etc/krb5.keytab file that gets created when a host joins an AD domain.

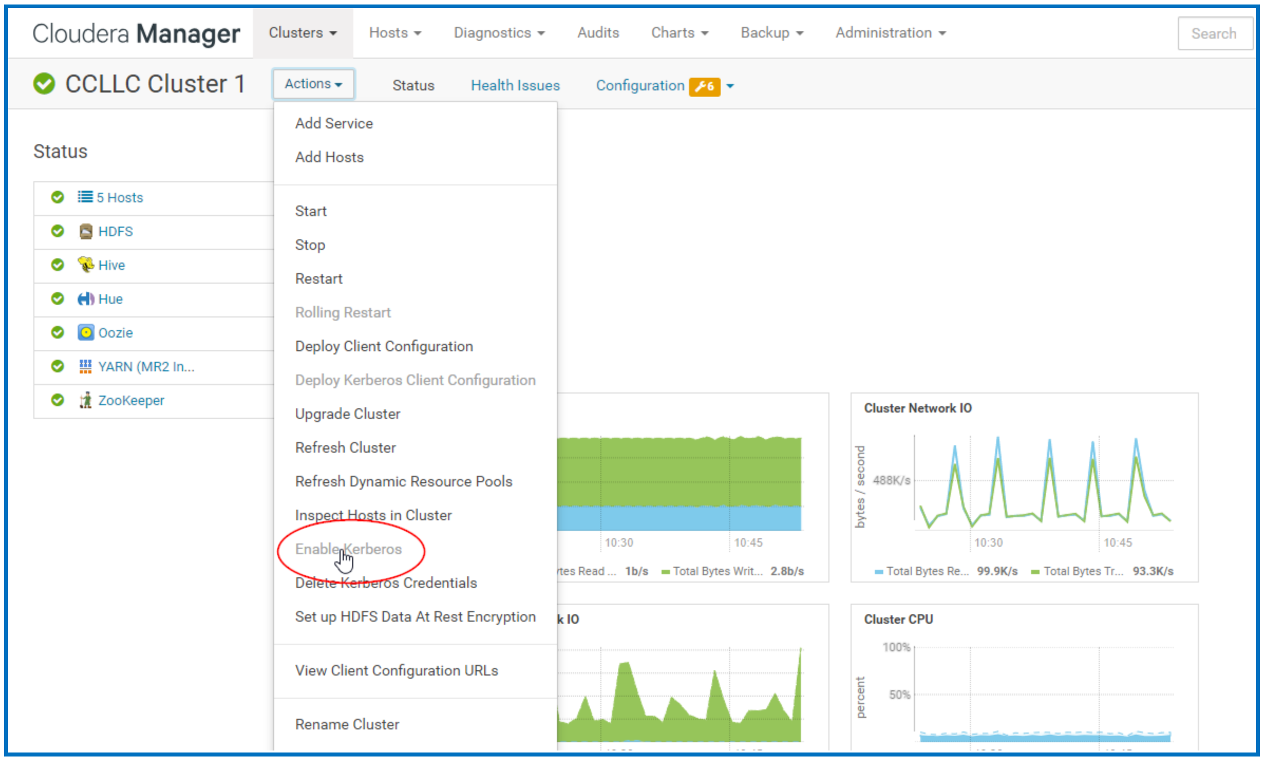
HADOOP KERBEROS CONFIGURATION

We've now reached the point where we will configure kerberized applications on our Linux hosts which are not part of the operating system distribution. These configurations will support Step 4 (where the TGS sends tickets to a user to enable communication with an application service) and Step 5 (where a user authenticates to the application service) of the Kerberos Authentication Example earlier in the paper.

The fundamental requirements to configure a kerberized application are:

1. A uniquely-named Kerberos Service Principal Name (SPN) for each service, including location information (e.g., hdfs/d1casprimary.ccllc.internal@CCLLC.INTERNAL, the SPN for the hdfs service on host d1casprimary.ccllc.internal).
2. In Active Directory (only), a User Principal (generally a service account) which the SPN above will be an attribute of. A dedicated User Principal is not needed with MIT Kerberos.
3. The keytab above stored locally on the application server.
4. The configuration of the application to activate Kerberos authentication, identify the application's Service Principal, and find its keytab.

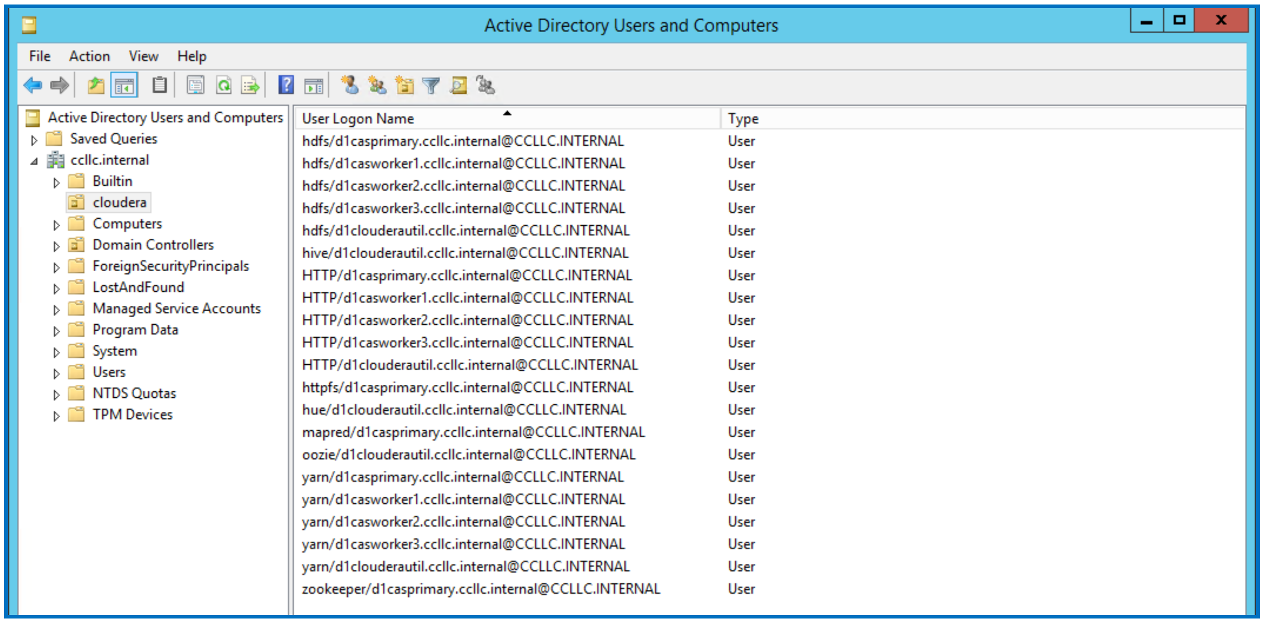
Cloudera 6.x Hadoop offers an "Easy Button" to kerberize an instance. After setting up a single user account in AD with the privileges to create new User Principals (or an equivalent admin account in MIT Kerberos), "Enable Kerberos" can be chosen from a pull-down menu in the Cloudera Manager Admin GUI (see Display 2).



Display 2. Cloudera Manager "Enable Kerberos" Option

The wizard will ask a few relevant questions, and then kerberize and restart the cluster.

Once the kerberization is complete, we can see that the wizard created about 20 User Principals in AD for us (see Display 3), which can be used to associate SPNs with. This satisfies the AD-Specific Requirement #2 above (this would not be necessary in MIT Kerberos).


Active Directory Users and Computers

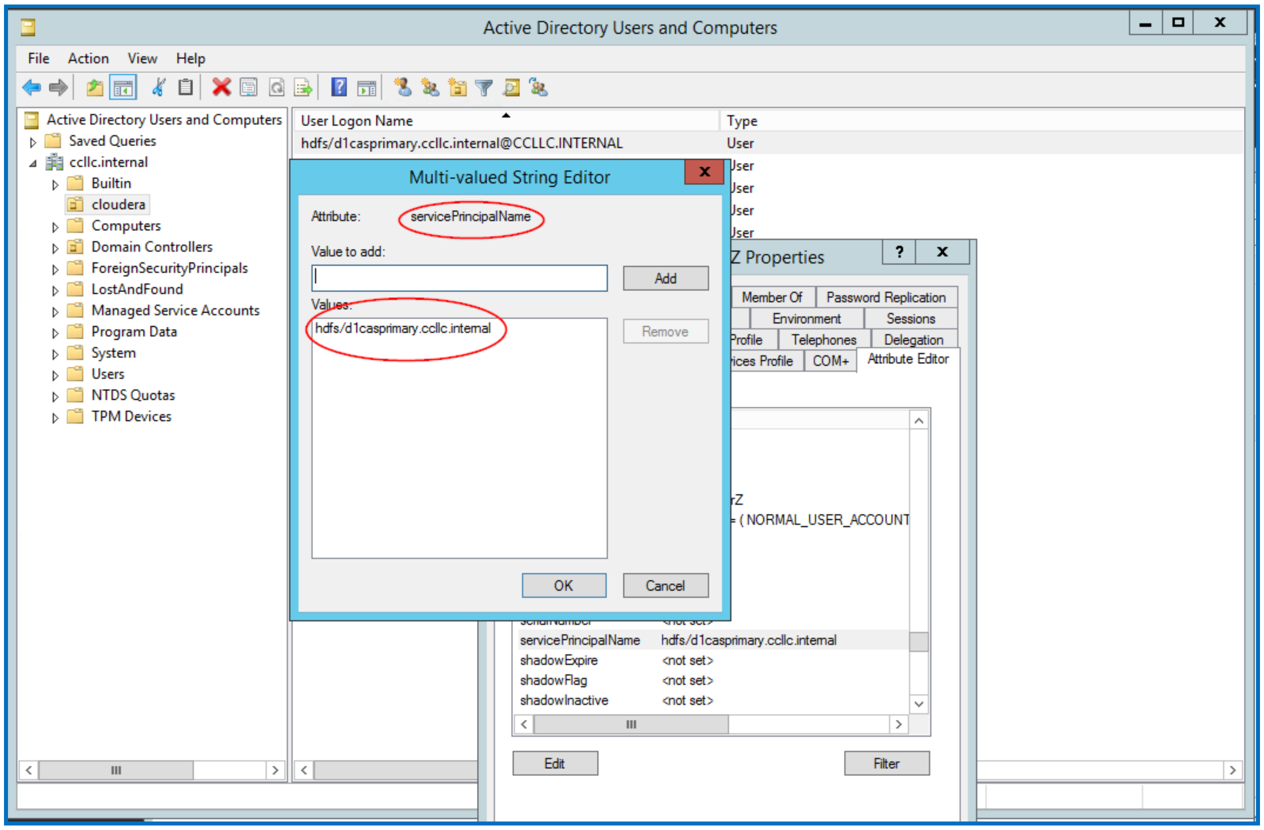
File Action View Help

- Active Directory Users and Computers
  - Saved Queries
  - ccllc.internal
    - Builtin
    - cloudera
    - Computers
    - Domain Controllers
    - ForeignSecurityPrincipals
    - LostAndFound
    - Managed Service Accounts
    - Program Data
    - System
    - Users
    - NTDS Quotas
    - TPM Devices

User Logon Name	Type
hdfs/d1casprimary.ccllc.internal@CCLLC.INTERNAL	User
hdfs/d1casworker1.ccllc.internal@CCLLC.INTERNAL	User
hdfs/d1casworker2.ccllc.internal@CCLLC.INTERNAL	User
hdfs/d1casworker3.ccllc.internal@CCLLC.INTERNAL	User
hdfs/d1clouderautil.ccllc.internal@CCLLC.INTERNAL	User
hive/d1clouderautil.ccllc.internal@CCLLC.INTERNAL	User
HTTP/d1casprimary.ccllc.internal@CCLLC.INTERNAL	User
HTTP/d1casworker1.ccllc.internal@CCLLC.INTERNAL	User
HTTP/d1casworker2.ccllc.internal@CCLLC.INTERNAL	User
HTTP/d1casworker3.ccllc.internal@CCLLC.INTERNAL	User
HTTP/d1clouderautil.ccllc.internal@CCLLC.INTERNAL	User
httpfs/d1casprimary.ccllc.internal@CCLLC.INTERNAL	User
hue/d1clouderautil.ccllc.internal@CCLLC.INTERNAL	User
mapred/d1casprimary.ccllc.internal@CCLLC.INTERNAL	User
oozie/d1clouderautil.ccllc.internal@CCLLC.INTERNAL	User
yarn/d1casprimary.ccllc.internal@CCLLC.INTERNAL	User
yarn/d1casworker1.ccllc.internal@CCLLC.INTERNAL	User
yarn/d1casworker2.ccllc.internal@CCLLC.INTERNAL	User
yarn/d1casworker3.ccllc.internal@CCLLC.INTERNAL	User
yarn/d1clouderautil.ccllc.internal@CCLLC.INTERNAL	User
zookeeper/d1casprimary.ccllc.internal@CCLLC.INTERNAL	User

Display 3. AD Service Account Users After Cloudera Kerberization

We can also see in AD that SPN's have been created and assigned as attributes of the new User Principals (see Display 4). (With MIT Kerberos, standalone Service Principals would be created.) This satisfies Requirement #1 above.


Active Directory Users and Computers

File Action View Help

- Active Directory Users and Computers
  - Saved Queries
  - ccllc.internal
    - Builtin
    - cloudera
    - Computers
    - Domain Controllers
    - ForeignSecurityPrincipals
    - LostAndFound
    - Managed Service Accounts
    - Program Data
    - System
    - Users
    - NTDS Quotas
    - TPM Devices

User Logon Name	Type
hdfs/d1casprimary.ccllc.internal@CCLLC.INTERNAL	User

Multi-valued String Editor

Attribute: servicePrincipalName

Value to add:

Values:

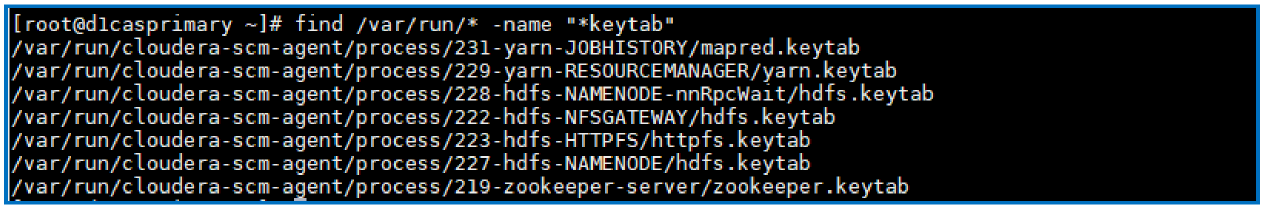
hdfs/d1casprimary.ccllc.internal

(NORMAL_USER_ACCOUNT

servicePrincipalName	hdfs/d1casprimary.ccllc.internal
shadowExpire	<not set>
shadowFlag	<not set>
shadowInactive	<not set>

Display 4. Hadoop Service Principal Attributes in AD

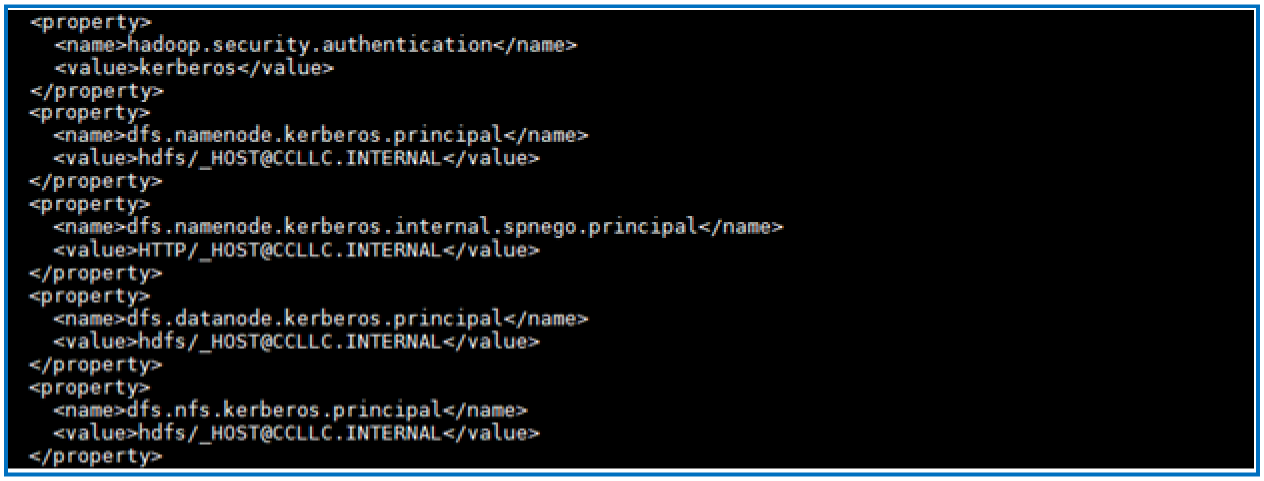
When the "Easy Button" path is chosen for Cloudera kerberization, the Cloudera Manager manages the keytabs on each Hadoop host. Output 29 shows the keytabs on the running Hadoop Namenode server right after kerberization. This satisfies Requirement #3 above.



```
[root@d1casprimary ~]# find /var/run/* -name "*keytab"
/var/run/cloudera-scm-agent/process/231-yarn-JOBHISTORY/mapred.keytab
/var/run/cloudera-scm-agent/process/229-yarn-RESOURCEMANAGER/yarn.keytab
/var/run/cloudera-scm-agent/process/228-hdfs-NAMENODE-nnRpcWait/hdfs.keytab
/var/run/cloudera-scm-agent/process/222-hdfs-NFSGATEWAY/hdfs.keytab
/var/run/cloudera-scm-agent/process/223-hdfs-HTTPFS/httpfs.keytab
/var/run/cloudera-scm-agent/process/227-hdfs-NAMENODE/hdfs.keytab
/var/run/cloudera-scm-agent/process/219-zookeeper-server/zookeeper.keytab
```

Output 29. Cloudera Hadoop Keytab Files

Finally, the application itself needs to be configured to support Kerberos. In the selected contents of the Hadoop *-site.xml configuration files shown in Output 30, we can see that Kerberos is enabled as the authentication method, and the Service Principal Names (SPNs) are identified. This satisfies the final Requirement #4 for a kerberized application.

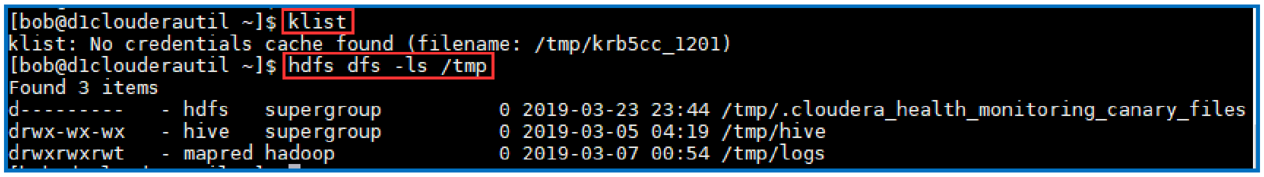


```
  <property>
    <name>hadoop.security.authentication</name>
    <value>kerberos</value>
  </property>
  <property>
    <name>dfs.namenode.kerberos.principal</name>
    <value>hdfs/_HOST@CCLLC.INTERNAL</value>
  </property>
  <property>
    <name>dfs.namenode.kerberos.internal.spnego.principal</name>
    <value>HTTP/_HOST@CCLLC.INTERNAL</value>
  </property>
  <property>
    <name>dfs.datanode.kerberos.principal</name>
    <value>hdfs/_HOST@CCLLC.INTERNAL</value>
  </property>
  <property>
    <name>dfs.nfs.kerberos.principal</name>
    <value>hdfs/_HOST@CCLLC.INTERNAL</value>
  </property>
```

Output 30. Selected Entries From Cloudera Hadoop *-site.xml Files Enabling Kerberos Support

One note from the screenshot above: you'll often see the word "spnego" when configuring kerberized applications accessible through a browser. SPNEGO stands for "Simple and Protected GSSAPI Negotiation Mechanism", but a SAS Administrator generally only needs to be aware that SPNEGO is used to support Kerberos over HTTP. In the screenshot above, for example, you can see that the Service Principal that will be used to support SPNEGO communications with the namenode will be "HTTP/_HOST@CCLLC.INTERNAL" ("_HOST" is a wildcard that will be replaced with a server's FQDN).

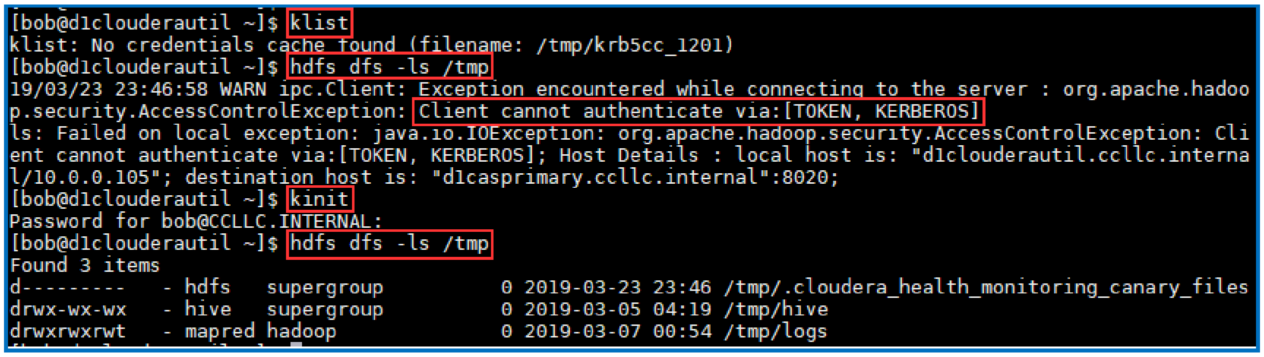
Before we kerberized our Hadoop cluster, anyone with a valid username on the server was able to execute Hadoop commands, as shown in Output 31.



```
[bob@d1clouderautil ~]$ klist
klist: No credentials cache found (filename: /tmp/krb5cc_1201)
[bob@d1clouderautil ~]$ hdfs dfs -ls /tmp
Found 3 items
d---------   - hdfs   supergroup          0 2019-03-23 23:44 /tmp/.cloudera_health_monitoring_canary_files
drwx-wx-wx   - hive   supergroup          0 2019-03-05 04:19 /tmp/hive
drwxrwxrwt   - mapred hadoop              0 2019-03-07 00:54 /tmp/logs
```

Output 31. HDFS Access Before Cloudera Hadoop Kerberization

But after kerberization, a valid Kerberos ticket is needed to access the Hadoop instance (see Output 32).



```
[bob@d1clouderautil ~]$ klist
klist: No credentials cache found (filename: /tmp/krb5cc_1201)
[bob@d1clouderautil ~]$ hdfs dfs -ls /tmp
19/03/23 23:46:58 WARN ipc.Client: Exception encountered while connecting to the server : org.apache.hadoo
p.security.AccessControlException: Client cannot authenticate via:[TOKEN, KERBEROS]
ls: Failed on local exception: java.io.IOException: org.apache.hadoop.security.AccessControlException: Cli
ent cannot authenticate via:[TOKEN, KERBEROS]; Host Details : local host is: "d1clouderautil.ccllc.interna
l/10.0.0.105"; destination host is: "d1casprimary.ccllc.internal":8020;
[bob@d1clouderautil ~]$ kinit
Password for bob@CCLLC.INTERNAL:
[bob@d1clouderautil ~]$ hdfs dfs -ls /tmp
Found 3 items
d---------   - hdfs   supergroup          0 2019-03-23 23:46 /tmp/.cloudera_health_monitoring_canary_files
drwx-wx-wx   - hive   supergroup          0 2019-03-05 04:19 /tmp/hive
drwxrwxrwt   - mapred hadoop              0 2019-03-07 00:54 /tmp/logs
```

Output 32. HDFS Access Only With Kerberos Credentials After Cloudera Hadoop Kerberization

SAS VIYA KERBEROS CONFIGURATION

There is no “easy button” yet in SAS Viya for configuring end-to-end Kerberos support including Single Sign-On through Integrated Windows Authentication (IWA), and therefore a number of manual steps will be needed for its implementation. However, it is important to realize that our system *as-is* can connect to downstream kerberized Hadoop instances with SAS Studio 4.x, and with very little additional configuration can connect to kerberized Hadoop with SAS Studio 5.x and CAS.

Our key reference material for the various Viya 3.4 authentication options below is Stuart Roger’s excellent blog post “SAS Viya 3.4 Authentication Overview”. In it and a group of follow-on blog posts related to Viya 3.4, he covers numerous possible SAS Viya authentication configurations in detail both with and without IWA support. We begin with SAS Viya without IWA support.

SAS VIYA WITHOUT IWA

If we do not need to implement IWA inbound to our SAS Viya environment, then with just a few changes we can enable comprehensive outbound connectivity to kerberized downstream datasources such as Hadoop from our SAS Viya environment.

SAS STUDIO 4.X

Upon login to SAS Studio 4.x, a SAS Object Spawner uses the host’s PAM configuration to validate the username and password, as shown in Figure 7.

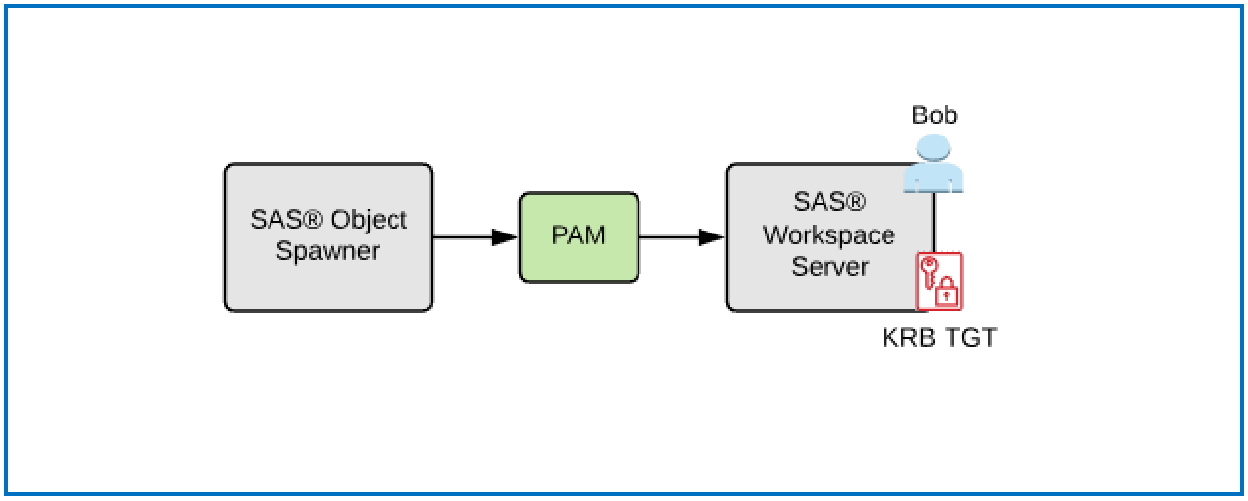
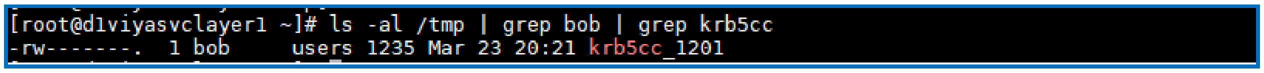


Figure 7. SAS Object Spawner PAM-Based Authentication

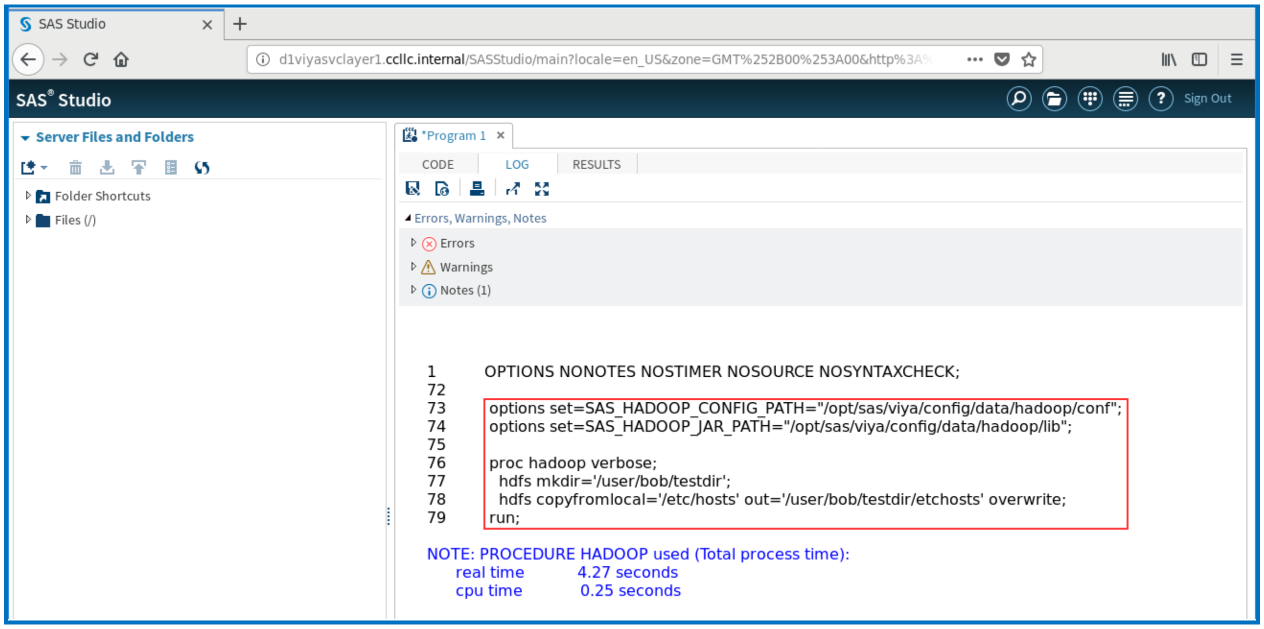
Since our environment's PAM configuration includes Kerberos authentication, a TGT will automatically be acquired for us upon SAS Studio 4.x login, as shown in Output 33.



```
[root@d1viyasvclayer1 ~]# ls -al /tmp | grep bob | grep krb5cc
-rw-------.  1 bob      users 1235 Mar 23 20:21 krb5cc_1201
```

Output 33. Filesystem Location Of The User's TGT After SASStudio 4.x Login

This TGT can then be used to access our kerberized Hadoop instance in our SAS Studio 4.x session without any additional changes, as shown in Display 5.



```
OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;

options set=SAS_HADOOP_CONFIG_PATH="/opt/sas/viya/config/data/hadoop/conf";
options set=SAS_HADOOP_JAR_PATH="/opt/sas/viya/config/data/hadoop/lib";

proc hadoop verbose;
  hdfs mkdir='/user/bob/testdir';
  hdfs copyfromlocal='/etc/hosts' out='/user/bob/testdir/etchosts' overwrite;
run;

NOTE: PROCEDURE HADOOP used (Total process time):
      real time           4.27 seconds
      cpu time            0.25 seconds
```

Display 5. Successful Access to Kerberized Hadoop In SASStudio 4.x

SAS STUDIO 5.X

SAS Studio 5.x follows a different instantiation and authentication path than SAS Studio 4.x. SAS Studio 5.x users authenticate against the SAS Logon Manager (instead of a SAS Object Spawner), and then the SAS Launcher Microservice and Server work together to launch an instance of a SAS Compute Server to host the SAS Studio 5.x session.

The critical piece of the flow for us to access downstream kerberized applications in IWA-less environments, is that the SAS Launcher Microservice will attempt to pull a user's stored credentials from the Credentials Microservice and use them in the process of launching the Compute Server. If it finds a set of credentials, the SAS Launcher Server will validate them through PAM, which will cause a Kerberos TGT to be created that can be used in our SAS Studio 5.x session (see Figure 8).

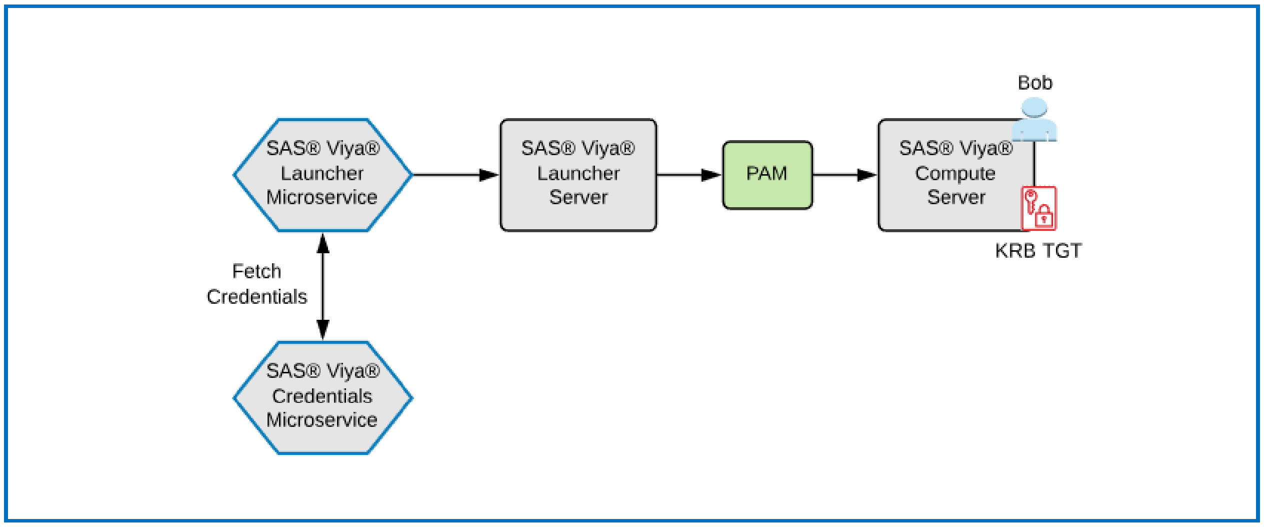
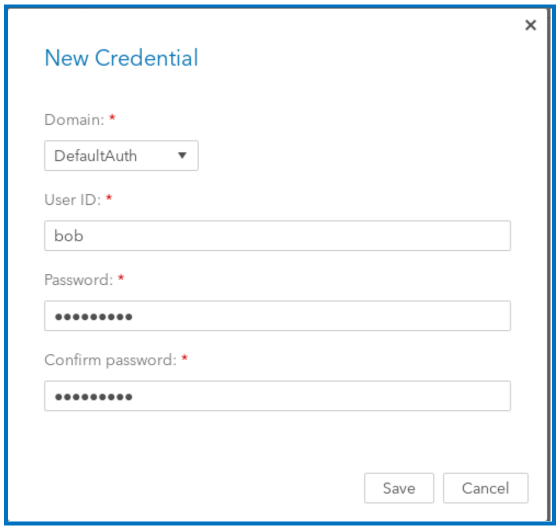

SAS® Viya® Launcher Microservice
SAS® Viya® Launcher Server
PAM
SAS® Viya® Compute Server
Bob
KRB TGT
Fetch Credentials
SAS® Viya® Credentials Microservice

Figure 8. SAS Viya Launcher Server PAM-Based Authentication

How do users store credentials in the Credentials Service in SAS Viya 3.4? Each user can log into SAS Viya Environment Manager as their own account, and choose the "My Credentials" key icon from the left-side menu. There they can add their username and password to the "DefaultAuth" domain, as shown in Display 6.

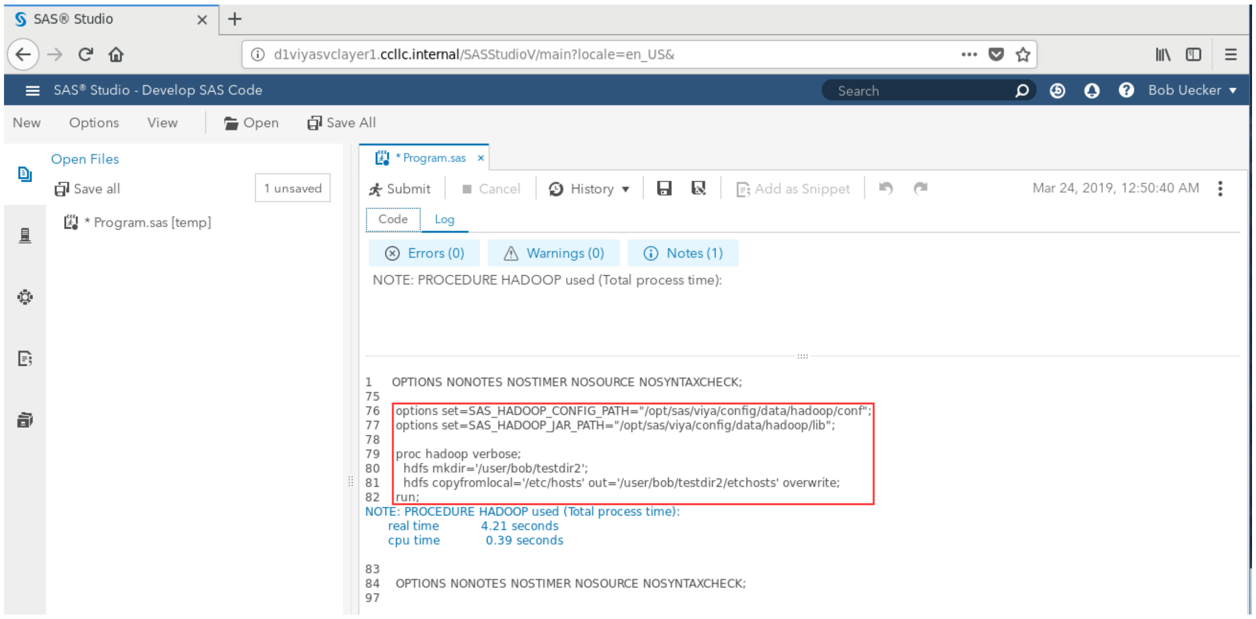


New Credential

Domain: *	DefaultAuth
User ID: *	bob
Password: *	•••••••••
Confirm password: *	•••••••••

Display 6. Storing User Credentials In SAS Viya Environment Manager

With our credentials stored in Environment Manager, we will then be able to have a Kerberos TGT available in our SAS Studio 5.x session, which can be used to access our kerberized Hadoop instance, as shown in Display 7.



```
NOTE: PROCEDURE HADOOP used (Total process time):

1    OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
75
76   options set=SAS_HADOOP_CONFIG_PATH="/opt/sas/viya/config/data/hadoop/conf";
77   options set=SAS_HADOOP_JAR_PATH="/opt/sas/viya/config/data/hadoop/lib";
78
79   proc hadoop verbose;
80     hdfs mkdir='/user/bob/testdir2';
81     hdfs copyfromlocal='/etc/hosts' out='/user/bob/testdir2/etchosts' overwrite;
82   run;
NOTE: PROCEDURE HADOOP used (Total process time):
      real time           4.21 seconds
      cpu time            0.39 seconds

83
84   OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
97
```

Display 7. Successful Access to Kerberized Hadoop In SASStudio 5.x

SAS CLOUD ANALYTIC SERVICES (CAS)

CAS authenticates and instantiates in a similar fashion to our SAS Studio 5.x flow. Users authenticate against the SAS Logon Manager, but this time the SAS CAS Controller launches a SAS CAS Session for the user.

The critical piece of the flow for us to access downstream kerberized applications in IWA-less environments, is that the SAS CAS Controller will attempt to pull a user's stored credentials from the Credentials Microservice and use them in the process of launching a SAS CAS Session for the user when the user is a member of the "CASHostAccountRequired" group. (Members of the "CASHostAccountRequired" group have their CAS sessions launched as their user account, instead of as the "cas" service account.) If it finds a set of credentials, the SAS CAS Controller will validate them through PAM, which will cause a Kerberos TGT to be created that can be used in our SAS CAS session (see Figure 9).

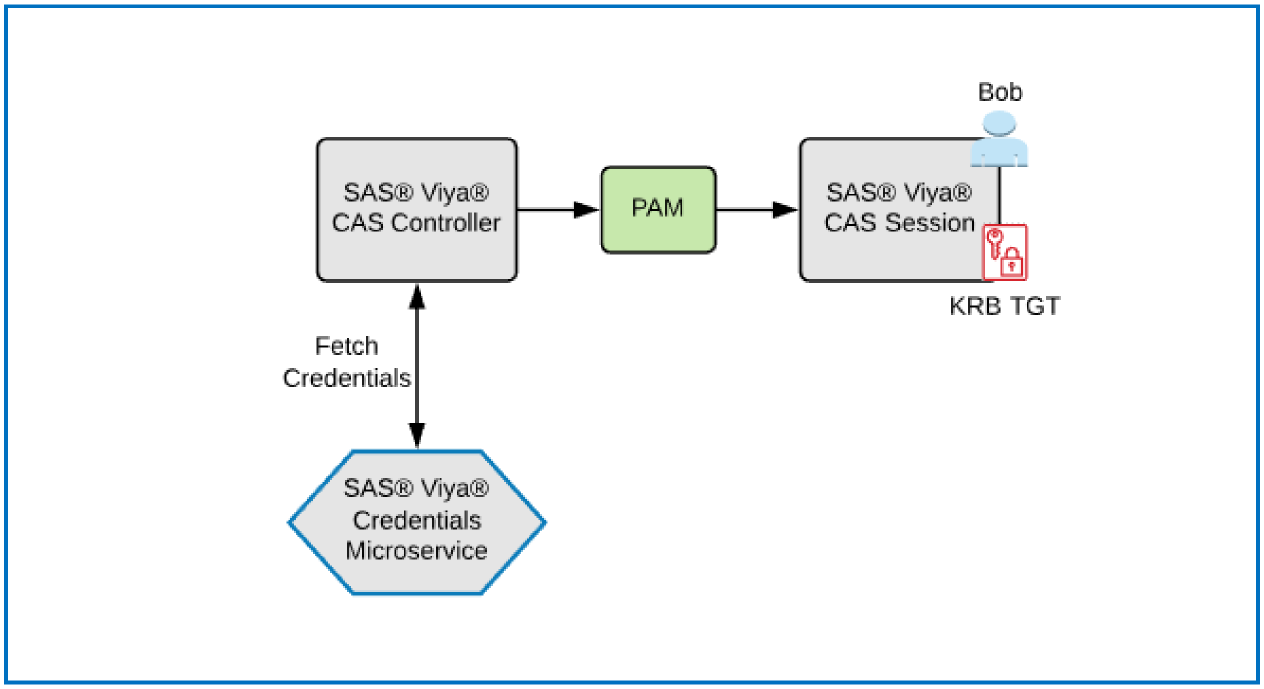
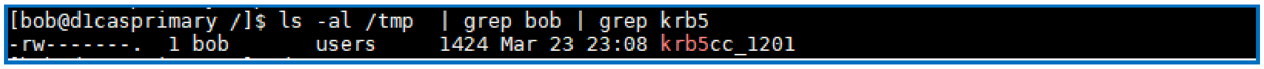

Bob
SAS® Viya® CAS Controller
PAM
SAS® Viya® CAS Session
KRB TGT
Fetch Credentials
SAS® Viya® Credentials Microservice

Figure 9 . SAS Viya CAS Server PAM-Based Authentication

If our credentials are stored in Environment Manager as described above, we will have a Kerberos TGT available in our CAS session, as shown in Output 34.

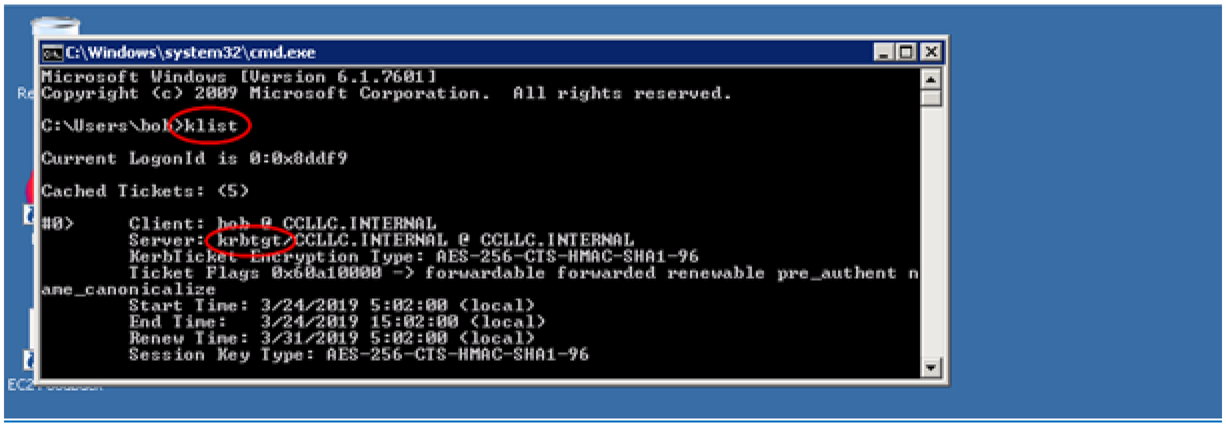

```
[bob@d1casprimary /]$ ls -al /tmp  | grep bob | grep krb5
-rw-------.  1 bob       users     1424 Mar 23 23:08 krb5cc_1201
```

Output 34. Filesystem Location Of The User's TGT After CAS Login

SAS VIYA WITH IWA

Fundamentals

Integrated Windows Authentication (IWA) is a technology on Windows platforms which allows users to access secured applications without a password prompt after their initial login to their workstation or laptop. IWA can use Kerberos authentication as one way to enable this seamless-access functionality. Because Microsoft implemented Kerberos as its default authentication method beginning with Windows 2000, when a user logs in to a computer which is a member of an AD Domain, a Kerberos TGT is automatically requested for the user (see Display 8). This TGT can then be used to access SAS Viya in a passwordless manner.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\bob>klist

Current LogonId is 0:0x8ddf9

Cached Tickets: (5)

#0>     Client: bob @ CCLLC.INTERNAL
        Server: krbtgt/CCLLC.INTERNAL @ CCLLC.INTERNAL
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x60a10000 -> forwardable forwarded renewable pre_authent n
ame_canonicalize
        Start Time: 3/24/2019 5:02:00 (local)
        End Time:   3/24/2019 15:02:00 (local)
        Renew Time: 3/31/2019 5:02:00 (local)
        Session Key Type: AES-256-CTS-HMAC-SHA1-96
```

Display 8. Kinit Output From An AD Domain Member Computer Immediately After Login

In order to implement full end-to-end Kerberos functionality in our SAS Viya environment, from front-end authentication inbound via IWA to back-end connectivity outbound to kerberized data sources like Hadoop, we need to kerberize a group of individual services in the SAS Viya architecture:

- **SAS Logon Manager:** To accept Kerberos credentials from the end-user's desktop
- **SAS Launcher Server:** To propagate Kerberos credentials into SAS Compute sessions
- **SAS CAS Server Controller:** To propagate Kerberos credentials into SAS CAS sessions

A key component of this functionality will be TGT *delegation*. In our SSH configuration above, we showed how Bob's TGT could be forwarded to another server upon login by setting the parameters "forwardable = true " and "GSSAPIDelegateCredentials on" in our /etc/krb5.conf and /etc/ssh/ssh_config, respectively. In our SAS Viya architecture, we are now going to allow our TGT to be delegated to the SAS Viya *Logon Manager*, for further delegation across the Viya ecosystem.

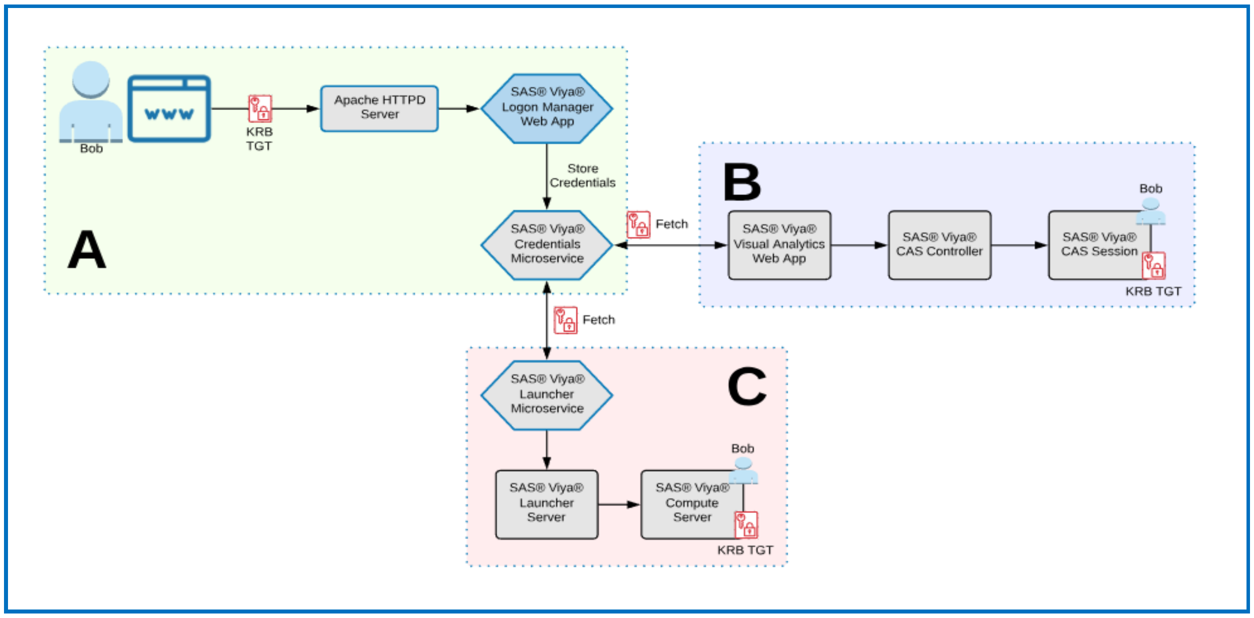

A
Bob
www
KRB TGT
Apache HTTPD Server
SAS® Viya® Logon Manager Web App
Store Credentials
SAS® Viya® Credentials Microservice
Fetch
B
SAS® Viya® Visual Analytics Web App
SAS® Viya® CAS Controller
SAS® Viya® CAS Session
Bob
KRB TGT
Fetch
C
SAS® Viya® Launcher Microservice
SAS® Viya® Launcher Server
SAS® Viya® Compute Server
Bob
KRB TGT

Figure 10. SAS Viya 3.4 Kerberos TGT Delegation Flows

The flows are as follows (shown in Figure 10):

A. Bob uses the Kerberos credentials on his Windows workstation to login passwordless to the SAS Logon Manager (this is Integrated Windows Authentication). Because Bob has a forwardable TGT and the SAS Logon Manager is "Trusted for Delegation" (see later in the paper), Bob's TGT will be forwarded to the SAS Logon Manager. The SAS Logon Manager will then store the Kerberos credentials in the Credentials Microservice.

B. When Bob wants to access CAS from an application such as Visual Analytics, VA will fetch Bob's Kerberos TGT from the Credentials Microservice, and use it to acquire a Service Ticket to communicate with the SAS CAS Controller (Step 3 in our earlier Kerberos Authentication Example). After authenticating to the SAS CAS Controller (Step 5 in the example), his TGT will be delegated again to the SAS CAS Controller, and then added to the credentials cache in the SAS CAS Session running under his own ID.

C. When Bob logs into SAS Studio 5.x and needs a Compute Server, the Launcher Microservice will fetch Bob's Kerberos TGT from the Credentials Microservice, and use it to acquire a Service Ticket to communicate with the SAS Launcher Server (Step 3 in our earlier Kerberos Authentication Example). After authenticating to the SAS Launcher Server (Step 5 in the example), his TGT will be delegated again to the SAS Launcher Server, and then added to the credentials cache in the SAS Compute Server running under his own ID.

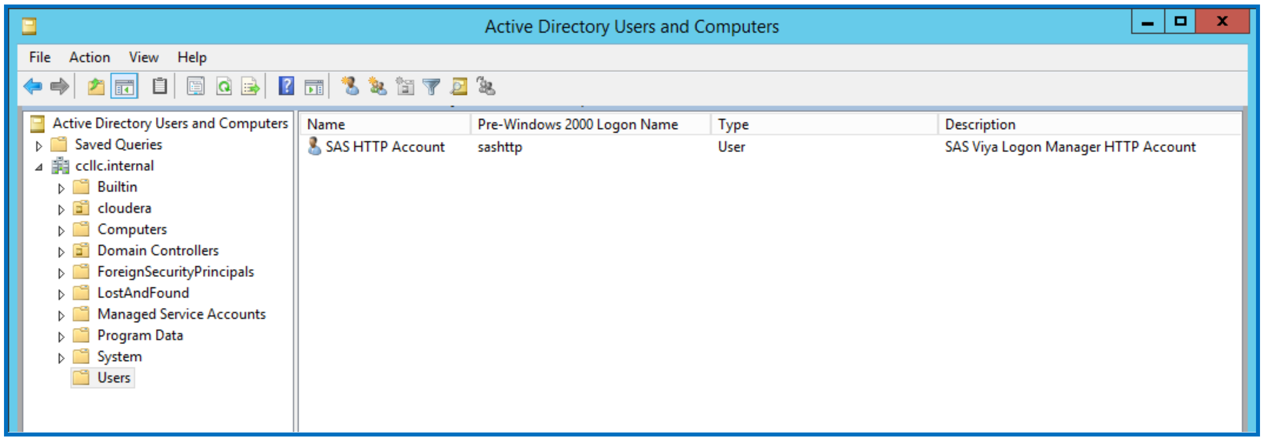
Service Kerberization

Recall the four fundamentals of kerberizing a service from our earlier Hadoop kerberization exercise:

1. A uniquely-named Kerberos Service Principal Name (SPN).
2. In Active Directory, a User Principal which the SPN above will be an attribute of.
3. A keytab stored locally on the application server.
4. The configuration of the application to accept Kerberos authentication, identify the application's Service Principal, and find its keytab.

We'll walk through the kerberization of the SAS Logon Manager.

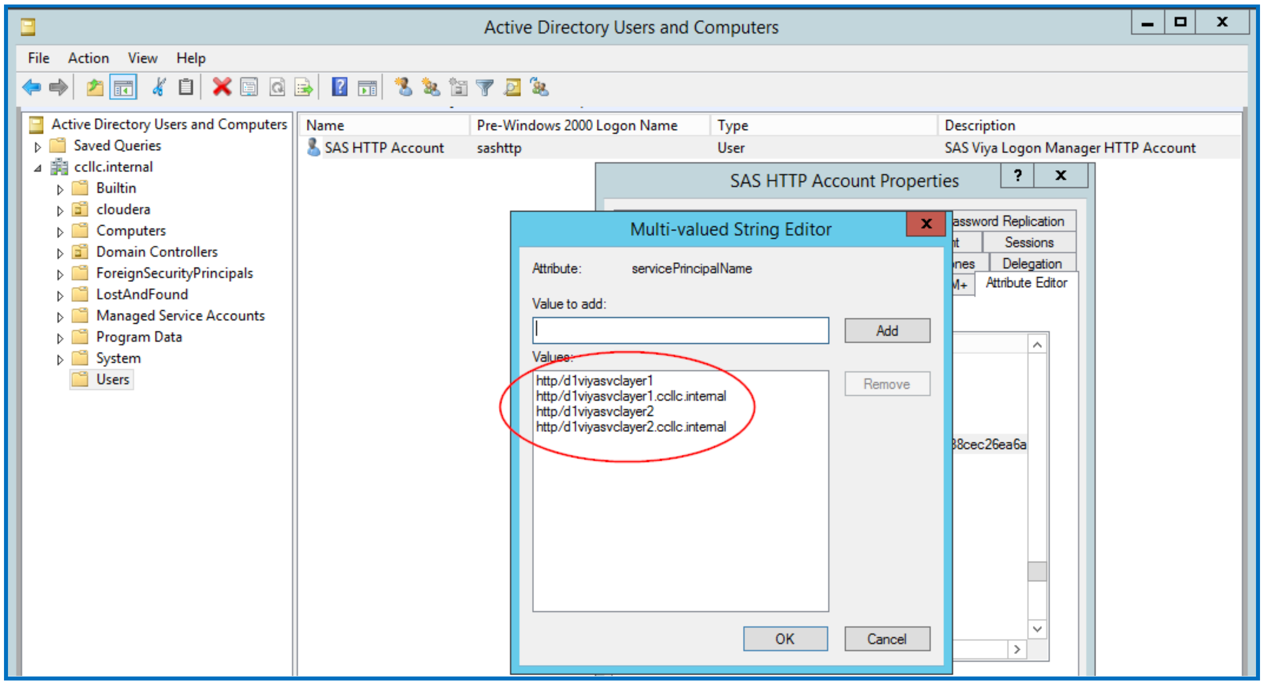
First we create a User Principal in Active Directory, as shown in Display 9. This satisfies AD-specific Requirement #2 (this would not be necessary in MIT Kerberos).



Name	Pre-Windows 2000 Logon Name	Type	Description
SAS HTTP Account	sashttp	User	SAS Viya Logon Manager HTTP Account

Display 9. The SAS Logon User Principal In AD

Then we need to add Service Principal Names as attributes of our User Principal (see Display 10). We add SPNs for our service (http, as SAS Logon Manager is accessed through a browser), with instance-names representing each server which runs the SAS Logon Manager, in both short-hostname and FQDN formats. (With MIT Kerberos, standalone Service Principals would be created.) This satisfies Requirement #1.



Name	Pre-Windows 2000 Logon Name	Type	Description
SAS HTTP Account	sashttp	User	SAS Viya Logon Manager HTTP Account

Attribute: servicePrincipalName

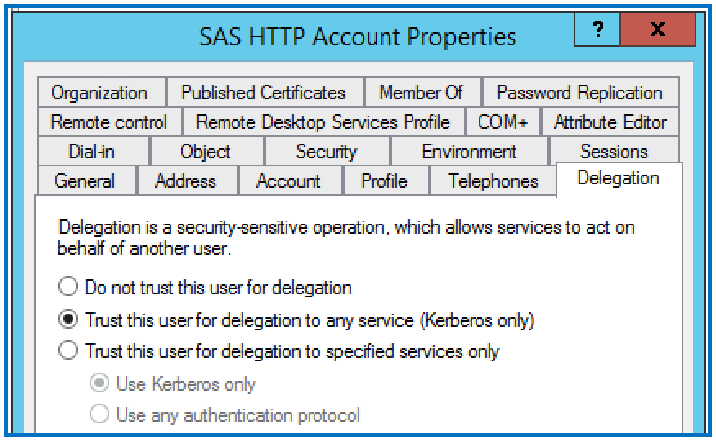
Value to add:

Values:

```
http/d1viyasvclayer1
http/d1viyasvclayer1.cclc.internal
http/d1viyasvclayer2
http/d1viyasvclayer2.cclc.internal
```

Display 10. The SAS Logon User Principal's Service Principal Attributes In AD

After we create our SPNs, we also need to enable "Trusted For Delegation" on the SAS Logon User Account (see Display 11), in order to support the SAS Viya TGT-delegation flow outlined above. Without this setting, Bob may authenticate to the Logon Manager, but his TGT will not be *forwarded* to the Logon Manager for storage in the Credentials Microservice.

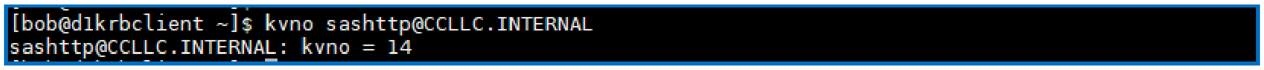


Delegation is a security-sensitive operation, which allows services to act on behalf of another user.

- ○ Do not trust this user for delegation
- ● Trust this user for delegation to any service (Kerberos only)
- ○ Trust this user for delegation to specified services only
  - ◉ Use Kerberos only
  - ○ Use any authentication protocol

Display 11. Trusted For Delegation Enablement In AD

We then need to create a keytab, and store it on the filesystem. On Linux, we can use the ktutil command to create the keytab. Before we run it, we need to determine the "kvno" (Key Version Number) of the User Principal which we will be creating a keytab for: this number is incremented each time a User Principal's password is changed. Acquire a TGT in order to be able to communicate with the KDC, and then run "kvno <username>", as shown in Output 35.



```
[bob@d1krbclient ~]$ kvno sashttp@CCLLC.INTERNAL
sashttp@CCLLC.INTERNAL: kvno = 14
```

Output 35. Determining A User Principal's Key Version Number With Kvno

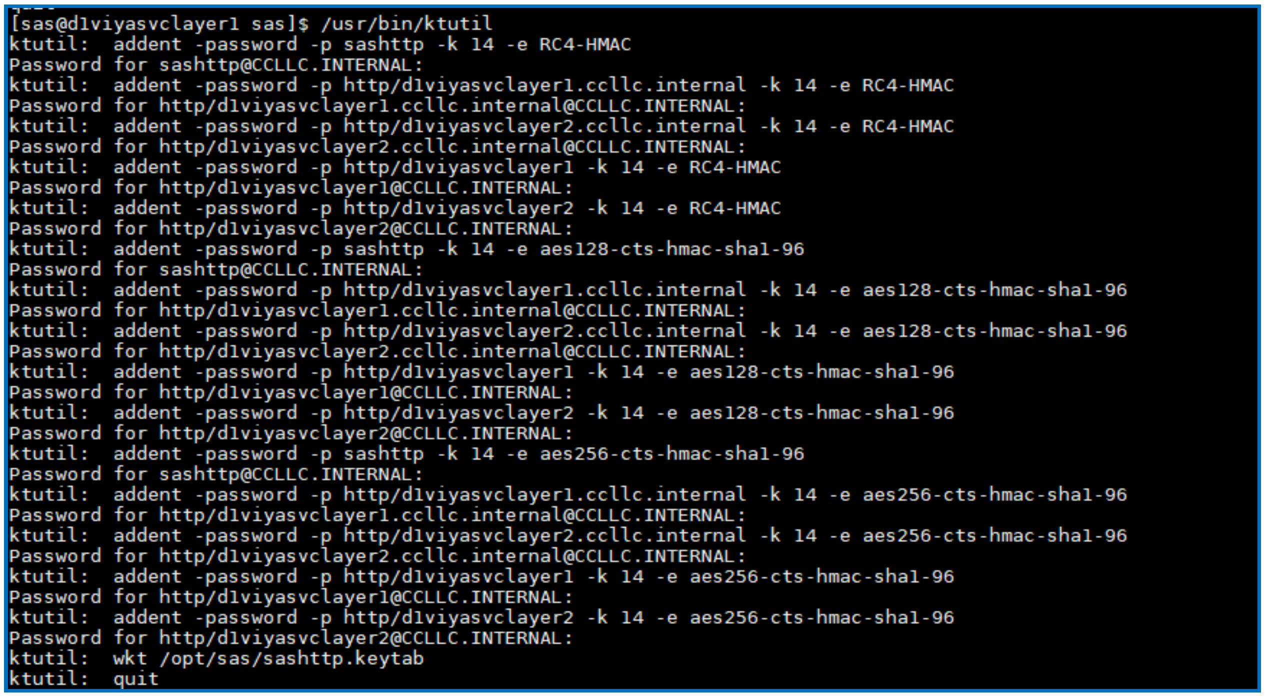
With the kvno in-hand, we will build a keytab using the following pattern:

```
ktutil
  addent password –p <UserPrincipal> -k <kvno> -e <encryption-type>
  addent password –p <ServicePrincipal> -k <kvno> -e <encryption-type>
  repeat for each UserPrincipal/ServicePrincipal/EncryptionType combo
  wkt <output-file>
  quit
```

Notes:

- After each "addent" command, you will be prompted for a password. The password for the User Principle associated with the keytab needs to be carefully entered. (With MIT Kerberos, the password for the Service Principal itself would be used.) The password is not being checked against the KDC, and an incorrect entry will cause the application service to not be able to decrypt the contents of Message H in Step 5 of our Kerberos Authorization Example, which will prevent users from being able to access the service.
- If you are unsure of the encryption types being used in your Kerberos environment, the most common three are RC4-HMAC, aes128-cts-hmac-sha1-96 and aes256-cts-hmac-sha1-96. Create entries for all three for each of the User and Service Principals.

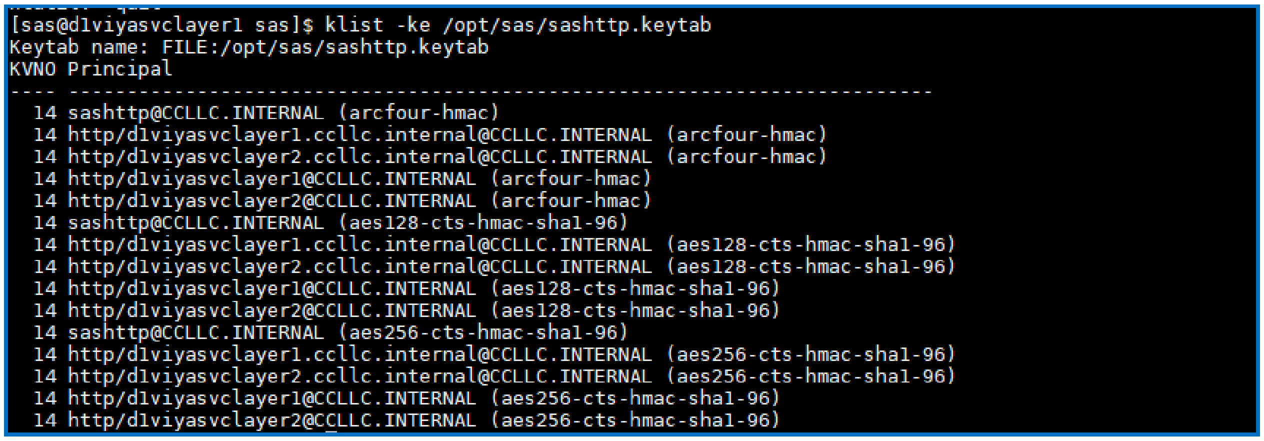
The ktutil command sequence to create our SAS Logon Service keytab is shown in Output 36.



```
[sas@d1viyasvclayer1 sas]$ /usr/bin/ktutil
ktutil:  addent -password -p sashttp -k 14 -e RC4-HMAC
Password for sashttp@CCLLC.INTERNAL:
ktutil:  addent -password -p http/d1viyasvclayer1.ccllc.internal -k 14 -e RC4-HMAC
Password for http/d1viyasvclayer1.ccllc.internal@CCLLC.INTERNAL:
ktutil:  addent -password -p http/d1viyasvclayer2.ccllc.internal -k 14 -e RC4-HMAC
Password for http/d1viyasvclayer2.ccllc.internal@CCLLC.INTERNAL:
ktutil:  addent -password -p http/d1viyasvclayer1 -k 14 -e RC4-HMAC
Password for http/d1viyasvclayer1@CCLLC.INTERNAL:
ktutil:  addent -password -p http/d1viyasvclayer2 -k 14 -e RC4-HMAC
Password for http/d1viyasvclayer2@CCLLC.INTERNAL:
ktutil:  addent -password -p sashttp -k 14 -e aes128-cts-hmac-sha1-96
Password for sashttp@CCLLC.INTERNAL:
ktutil:  addent -password -p http/d1viyasvclayer1.ccllc.internal -k 14 -e aes128-cts-hmac-sha1-96
Password for http/d1viyasvclayer1.ccllc.internal@CCLLC.INTERNAL:
ktutil:  addent -password -p http/d1viyasvclayer2.ccllc.internal -k 14 -e aes128-cts-hmac-sha1-96
Password for http/d1viyasvclayer2.ccllc.internal@CCLLC.INTERNAL:
ktutil:  addent -password -p http/d1viyasvclayer1 -k 14 -e aes128-cts-hmac-sha1-96
Password for http/d1viyasvclayer1@CCLLC.INTERNAL:
ktutil:  addent -password -p http/d1viyasvclayer2 -k 14 -e aes128-cts-hmac-sha1-96
Password for http/d1viyasvclayer2@CCLLC.INTERNAL:
ktutil:  addent -password -p sashttp -k 14 -e aes256-cts-hmac-sha1-96
Password for sashttp@CCLLC.INTERNAL:
ktutil:  addent -password -p http/d1viyasvclayer1.ccllc.internal -k 14 -e aes256-cts-hmac-sha1-96
Password for http/d1viyasvclayer1.ccllc.internal@CCLLC.INTERNAL:
ktutil:  addent -password -p http/d1viyasvclayer2.ccllc.internal -k 14 -e aes256-cts-hmac-sha1-96
Password for http/d1viyasvclayer2.ccllc.internal@CCLLC.INTERNAL:
ktutil:  addent -password -p http/d1viyasvclayer1 -k 14 -e aes256-cts-hmac-sha1-96
Password for http/d1viyasvclayer1@CCLLC.INTERNAL:
ktutil:  addent -password -p http/d1viyasvclayer2 -k 14 -e aes256-cts-hmac-sha1-96
Password for http/d1viyasvclayer2@CCLLC.INTERNAL:
ktutil:  wkt /opt/sas/sashttp.keytab
ktutil:  quit
```

Output 36. Creating A Keytab With Ktutil

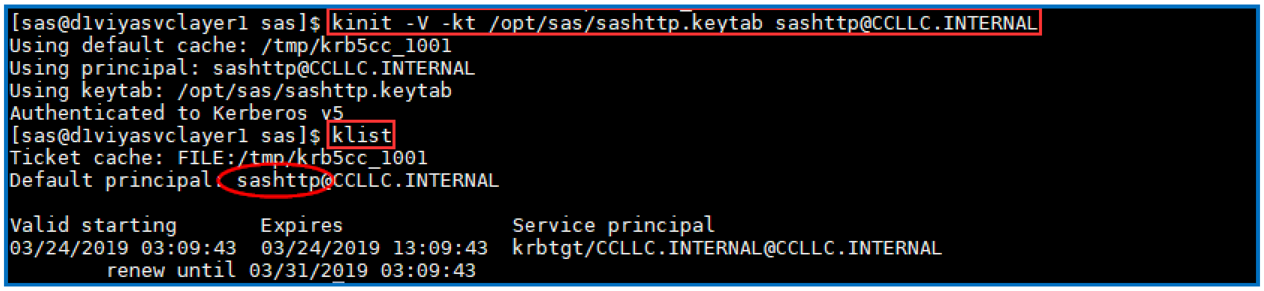
Once the keytab has been created, the contents can be listed for validation with klist, as shown in Output 37.



```
[sas@d1viyasvclayer1 sas]$ klist -ke /opt/sas/sashttp.keytab
Keytab name: FILE:/opt/sas/sashttp.keytab
KVNO Principal
---- --------------------------------------------------------------------------
  14 sashttp@CCLLC.INTERNAL (arcfour-hmac)
  14 http/d1viyasvclayer1.ccllc.internal@CCLLC.INTERNAL (arcfour-hmac)
  14 http/d1viyasvclayer2.ccllc.internal@CCLLC.INTERNAL (arcfour-hmac)
  14 http/d1viyasvclayer1@CCLLC.INTERNAL (arcfour-hmac)
  14 http/d1viyasvclayer2@CCLLC.INTERNAL (arcfour-hmac)
  14 sashttp@CCLLC.INTERNAL (aes128-cts-hmac-sha1-96)
  14 http/d1viyasvclayer1.ccllc.internal@CCLLC.INTERNAL (aes128-cts-hmac-sha1-96)
  14 http/d1viyasvclayer2.ccllc.internal@CCLLC.INTERNAL (aes128-cts-hmac-sha1-96)
  14 http/d1viyasvclayer1@CCLLC.INTERNAL (aes128-cts-hmac-sha1-96)
  14 http/d1viyasvclayer2@CCLLC.INTERNAL (aes128-cts-hmac-sha1-96)
  14 sashttp@CCLLC.INTERNAL (aes256-cts-hmac-sha1-96)
  14 http/d1viyasvclayer1.ccllc.internal@CCLLC.INTERNAL (aes256-cts-hmac-sha1-96)
  14 http/d1viyasvclayer2.ccllc.internal@CCLLC.INTERNAL (aes256-cts-hmac-sha1-96)
  14 http/d1viyasvclayer1@CCLLC.INTERNAL (aes256-cts-hmac-sha1-96)
  14 http/d1viyasvclayer2@CCLLC.INTERNAL (aes256-cts-hmac-sha1-96)
```

Output 37. Displaying The Contents Of A Keytab With Klist

The keytab’s ability to acquire a TGT *without needing a password prompt* can then be tested for the User Principal in AD (or the Service Principal in MIT Kerberos), as shown in Output 38.



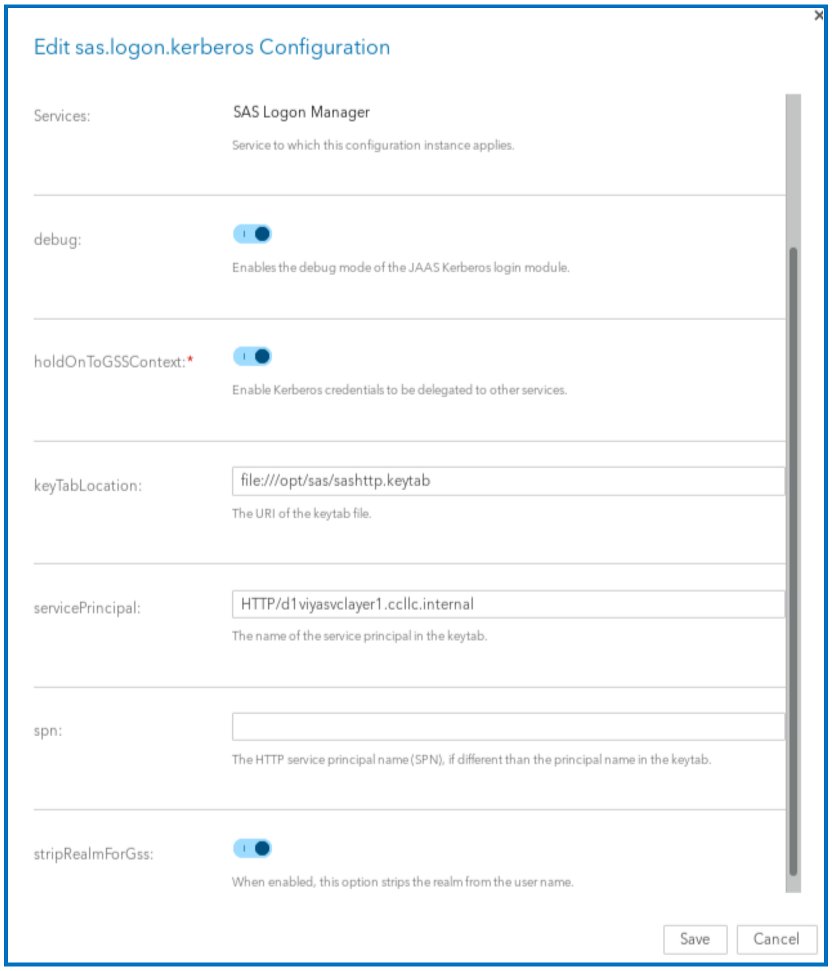
```
[sas@d1viyasvclayer1 sas]$ kinit -V -kt /opt/sas/sashttp.keytab sashttp@CCLLC.INTERNAL
Using default cache: /tmp/krb5cc_1001
Using principal: sashttp@CCLLC.INTERNAL
Using keytab: /opt/sas/sashttp.keytab
Authenticated to Kerberos v5
[sas@d1viyasvclayer1 sas]$ klist
Ticket cache: FILE:/tmp/krb5cc_1001
Default principal: sashttp@CCLLC.INTERNAL

Valid starting       Expires              Service principal
03/24/2019 03:09:43  03/24/2019 13:09:43  krbtgt/CCLLC.INTERNAL@CCLLC.INTERNAL
        renew until 03/31/2019 03:09:43
```

Output 38. Acquiring A TGT Passwordless Using A Keytab

By storing the keytab on the local filesystem and making it readable only by the sas account which runs the SAS Logon Manager process, Requirement #3 above is met.

Finally, we need to configure the application to support Kerberos authentication, identify the application's Service Principal, and find its keytab. This will allow us to meet kerberization Requirement #4 above. We'll do that on a pair of screens in SAS Viya 3.4's Environment Manager as an Administrator.

One screen looks like Display 12.



Edit sas.logon.kerberos Configuration

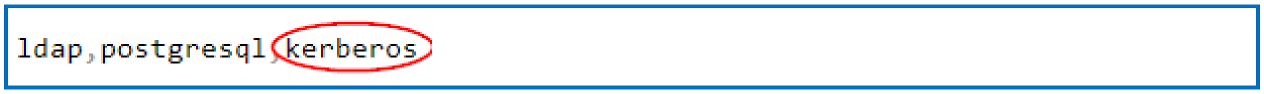
Services:	SAS Logon Manager Service to which this configuration instance applies.
debug:	● (on) Enables the debug mode of the JAAS Kerberos login module.
holdOnToGSSContext:*	● (on) Enable Kerberos credentials to be delegated to other services.
keyTabLocation:	file:///opt/sas/sashttp.keytab The URI of the keytab file.
servicePrincipal:	HTTP/d1viyasvclayer1.ccllc.internal The name of the service principal in the keytab.
spn:	 The HTTP service principal name (SPN), if different than the principal name in the keytab.
stripRealmForGss:	● (on) When enabled, this option strips the realm from the user name.

Display 12. SAS Logon Manager Kerberos Configuration In SAS Environment Manager

Two non-intuitive entries:

- **holdOnToGSSContext:** Tells the SAS Logon Manager to store the delegated user TGT in the Credentials Microservice, for later use with the CAS Controller and Launcher Servers
- **stripRealmForGSS:** Removes the "@REALM" from the end of usernames for OS-layer activities

The other Environment Manager screen simply adds Kerberos as an available authentication option under the "spring" settings of the SAS Logon Manager (see Display 13).



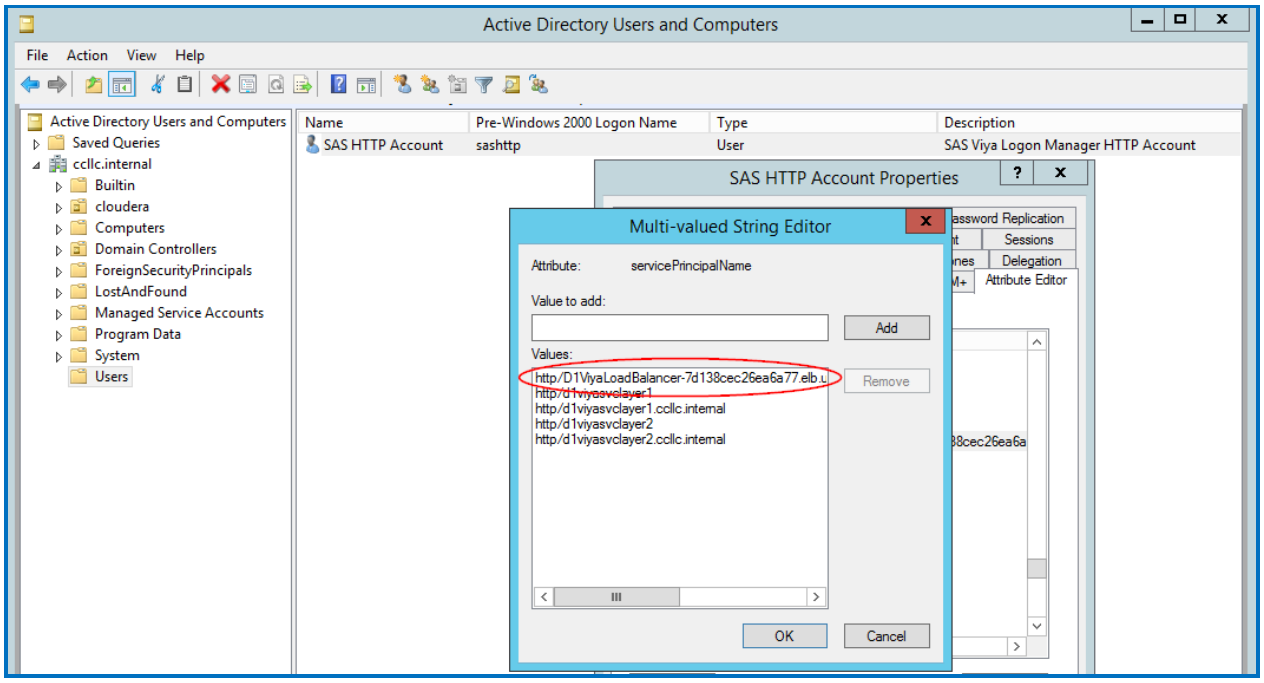
```
ldap,postgresql,kerberos
```

Display 13. SAS Logon Manager Kerberos Enablement In SAS Environment Manager

With a restart of the SAS Logon Manager, it will now be kerberized and able to validate Kerberos service tickets for authentication. The kerberization of the SAS CAS Controller and SAS Launcher Server follow the same *fundamentals*, but with slightly different steps as outlined in the SAS Viya 3.4 Administration Guide.

Load Balancers

You may have noticed above that our environment has two servers hosting SAS Logon Services (d1viyasvclayer1 and d1viyasvclayer2). If we have a load balancer in front of our multiple SAS Viya Service Layer servers, support for Kerberos through the load balancer is very straightforward: we just need a Service Principal Name for our Load Balancer address, as shown in Display 14.



Name	Pre-Windows 2000 Logon Name	Type	Description
SAS HTTP Account	sashttp	User	SAS Viya Logon Manager HTTP Account

Attribute: servicePrincipalName

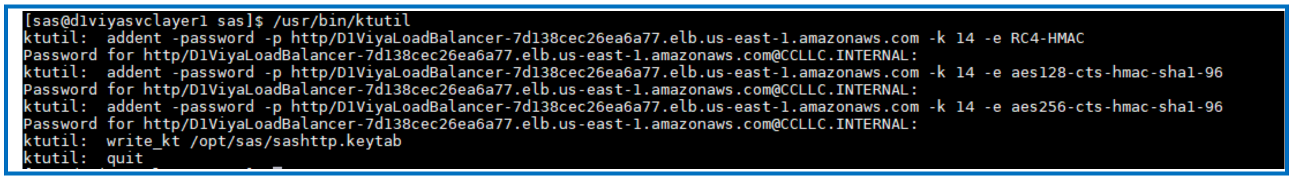
Value to add:

Values:

```
http/D1ViyaLoadBalancer-7d138cec26ea6a77.elb.u
http/d1viyasvclayer1
http/d1viyasvclayer1.ccllc.internal
http/d1viyasvclayer2
http/d1viyasvclayer2.ccllc.internal
```

Display 14. Adding A Load Balancer Address As Service Principal Attribute In AD

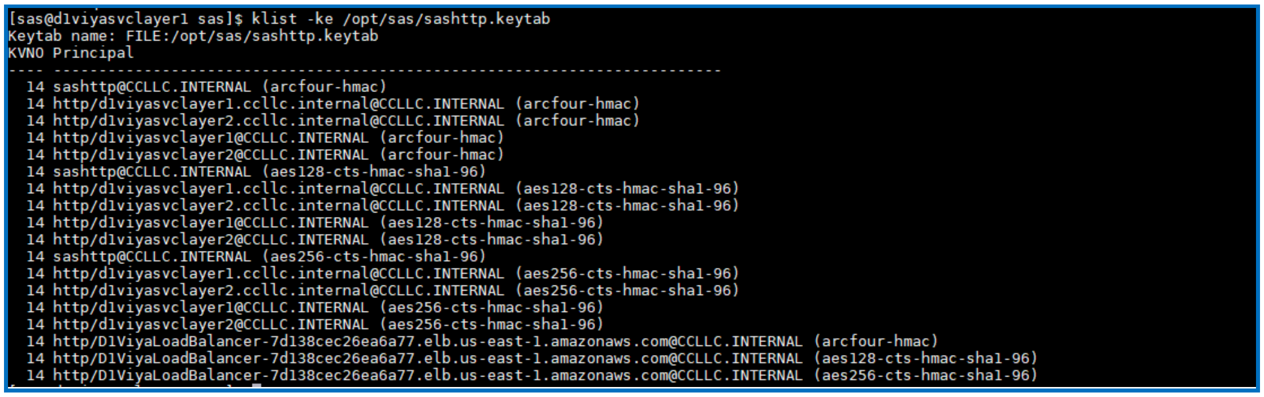
We can then add the SPN to our Kerberos keytab with ktutil, as shown in Output 39.



```
[sas@d1viyasvclayer1 sas]$ /usr/bin/ktutil
ktutil:  addent -password -p http/D1ViyaLoadBalancer-7d138cec26ea6a77.elb.us-east-1.amazonaws.com -k 14 -e RC4-HMAC
Password for http/D1ViyaLoadBalancer-7d138cec26ea6a77.elb.us-east-1.amazonaws.com@CCLLC.INTERNAL:
ktutil:  addent -password -p http/D1ViyaLoadBalancer-7d138cec26ea6a77.elb.us-east-1.amazonaws.com -k 14 -e aes128-cts-hmac-sha1-96
Password for http/D1ViyaLoadBalancer-7d138cec26ea6a77.elb.us-east-1.amazonaws.com@CCLLC.INTERNAL:
ktutil:  addent -password -p http/D1ViyaLoadBalancer-7d138cec26ea6a77.elb.us-east-1.amazonaws.com -k 14 -e aes256-cts-hmac-sha1-96
Password for http/D1ViyaLoadBalancer-7d138cec26ea6a77.elb.us-east-1.amazonaws.com@CCLLC.INTERNAL:
ktutil:  write_kt /opt/sas/sashttp.keytab
ktutil:  quit
```

Output 39. Adding A Load Balancer Address To A Keytab With Ktutil

We can then validate that our entries are available in our klist, as shown in Output 40.



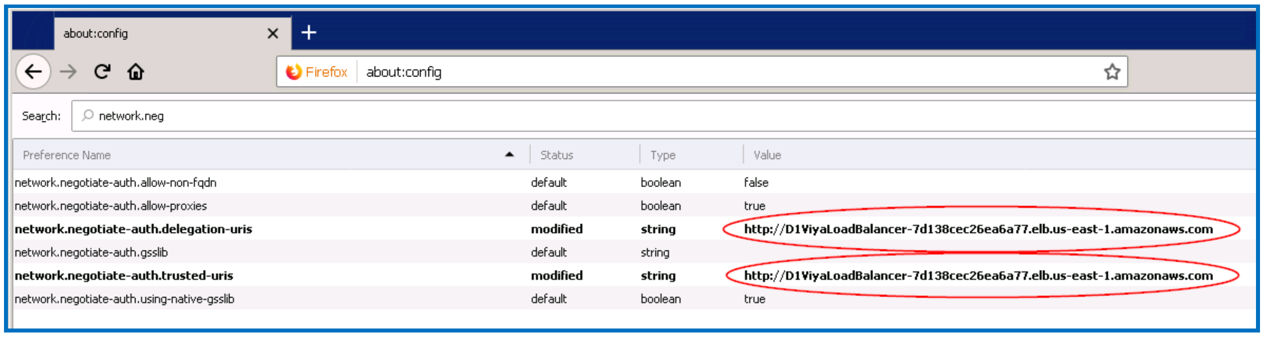
```
[sas@d1viyasvclayer1 sas]$ klist -ke /opt/sas/sashttp.keytab
Keytab name: FILE:/opt/sas/sashttp.keytab
KVNO Principal
---- --------------------------------------------------------------------------
  14 sashttp@CCLLC.INTERNAL (arcfour-hmac)
  14 http/d1viyasvclayer1.ccllc.internal@CCLLC.INTERNAL (arcfour-hmac)
  14 http/d1viyasvclayer2.ccllc.internal@CCLLC.INTERNAL (arcfour-hmac)
  14 http/d1viyasvclayer1@CCLLC.INTERNAL (arcfour-hmac)
  14 http/d1viyasvclayer2@CCLLC.INTERNAL (arcfour-hmac)
  14 sashttp@CCLLC.INTERNAL (aes128-cts-hmac-sha1-96)
  14 http/d1viyasvclayer1.ccllc.internal@CCLLC.INTERNAL (aes128-cts-hmac-sha1-96)
  14 http/d1viyasvclayer2.ccllc.internal@CCLLC.INTERNAL (aes128-cts-hmac-sha1-96)
  14 http/d1viyasvclayer1@CCLLC.INTERNAL (aes128-cts-hmac-sha1-96)
  14 http/d1viyasvclayer2@CCLLC.INTERNAL (aes128-cts-hmac-sha1-96)
  14 sashttp@CCLLC.INTERNAL (aes256-cts-hmac-sha1-96)
  14 http/d1viyasvclayer1.ccllc.internal@CCLLC.INTERNAL (aes256-cts-hmac-sha1-96)
  14 http/d1viyasvclayer2.ccllc.internal@CCLLC.INTERNAL (aes256-cts-hmac-sha1-96)
  14 http/d1viyasvclayer1@CCLLC.INTERNAL (aes256-cts-hmac-sha1-96)
  14 http/d1viyasvclayer2@CCLLC.INTERNAL (aes256-cts-hmac-sha1-96)
  14 http/D1ViyaLoadBalancer-7d138cec26ea6a77.elb.us-east-1.amazonaws.com@CCLLC.INTERNAL (arcfour-hmac)
  14 http/D1ViyaLoadBalancer-7d138cec26ea6a77.elb.us-east-1.amazonaws.com@CCLLC.INTERNAL (aes128-cts-hmac-sha1-96)
  14 http/D1ViyaLoadBalancer-7d138cec26ea6a77.elb.us-east-1.amazonaws.com@CCLLC.INTERNAL (aes256-cts-hmac-sha1-96)
```

Output 40. Displaying The Contents Of A Keytab With A Load Balancer Using Klist

DESKTOP IWA CONFIGURATION

Each major browser has a bit different configuration to enable IWA. We'll use Firefox as an example.

In about:config there are two Kerberos-related settings that need the hostnames and/or aliases of the SAS Logon webservers added, as shown in Display 15.



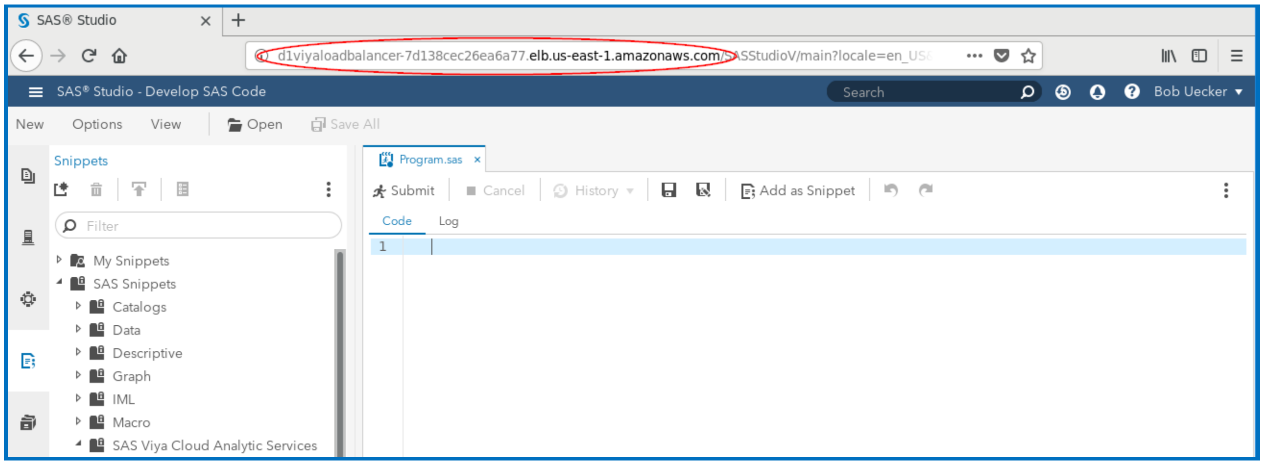
Preference Name	Status	Type	Value
network.negotiate-auth.allow-non-fqdn	default	boolean	false
network.negotiate-auth.allow-proxies	default	boolean	true
network.negotiate-auth.delegation-uris	modified	string	http://D1ViyaLoadBalancer-7d138cec26ea6a77.elb.us-east-1.amazonaws.com
network.negotiate-auth.gsslib	default	string	
network.negotiate-auth.trusted-uris	modified	string	http://D1ViyaLoadBalancer-7d138cec26ea6a77.elb.us-east-1.amazonaws.com
network.negotiate-auth.using-native-gsslib	default	boolean	true

Display 15. Firefox Configuration For IWA Access To The Load Balancer URL

- **network.negotiate-auth.trusted-urls:** The list of hosts that Firefox will attempt to authenticate with using Kerberos
- **network.negotiate-auth.delegation-urls:** The list of hosts that Firefox will delegate our TGT to if asked

Once these two settings are in-place, and our SAS/Hadoop environments are fully kerberized, we can use Firefox to log in to our SAS Viya visual applications (e.g., Visual

Analytics, SAS Studio 5.x) without a password check, and have full outbound access to kerberized downstream datasources also without a password check (see Display 16).



Display 16. SASStudio 5.x Access Through A Load Balancer With IWA

CONCLUSION

Kerberos is an enterprise-grade network authentication protocol which can significantly improve application and data security in a corporate environment. However the complexity of the protocol, along with the variety of ways it can be implemented and configured, can make it difficult for SAS administrators to fully understand the entire SAS and Kerberos integration picture.

We hope that this paper, where we took the approach of systematically explaining the key configurations that are common across all kerberized application deployments instead of focusing on the granular steps tied to a particular revision of the OS or application, has been helpful towards better understanding the deployment and integration options in your own corporate environment, and improving your Kerberos troubleshooting skills.

REFERENCES

Rogers, Stuart. "SAS Viya 3.4 Authentication Overview." Available at https://communities.sas.com/t5/SAS-Communities-Library/SAS-Viya-3-4-Authentication-Overview/ta-p/523244.

Pal, Dmitri. "Overview of Direct Integration Options." Available at https://rhelblog.redhat.com/2015/02/04/overview-of-direct-integration-options/.

Microsoft. "How the Kerberos Version 5 Authentication Protocol Works." Available at https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc772815(v%3dws.10).

RedHat. "Red Hat Enterprise Linux 7 System-Level Authentication Guide." Available at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/system-level_authentication_guide/index.

Cloudera. "Enabling Kerberos Authentication for CDH." Available at https://www.cloudera.com/documentation/enterprise/6/6.1/topics/cm_sg_intro_kerb.html.

SAS. "SAS Viya 3.4 Administration Guide." Available at https://documentation.sas.com/?cdcId=calcdc&cdcVersion=3.4&docsetId=calchkcfg&docsetTarget=n00004saschecklist0000config.htm&locale=en.

SAS. "SAS Viya 3.4 for Linux: Deployment Guide." Available at https://documentation.sas.com/?docsetId=dplyml0phy0lax&docsetTarget=titlepage.htm&docsetVersion=3.4&locale=en.

ACKNOWLEDGMENTS

We would like to thank Stuart Rogers for his excellent work in developing best practices and documentation in the area of SAS, Hadoop and Kerberos integrations. It has been invaluable in supporting our SAS clients with kerberized environments.

We would also like to thank Carlos Phillips, Don Hayes, and Rebecca Hayes for reviewing this paper and offering valuable suggestions.

RECOMMENDED READING

Root, Lynn. "Explain like I'm 5 Kerberos." Available at http://www.roguelynn.com/words/explain-like-im-5-kerberos.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Michael Shealy
Cached Consulting LLC
michael.shealy@cachedconsulting.com

Spencer Hayes
Cached Consulting LLC
spencer.hayes@cachedconsulting.com

www.cachedconsulting.com