Single Sign-On Using Kerberos

Chris Eberle, Ryan Thomas, RC Johnson, and Kim-Lan Tran

CS-591 Fall 2008

1. Introduction

In most modern networks you can count on two things: services and users. Services are typically things like email, web sites, and shell accounts. Traditionally, each service on a network had been viewed as an autonomous unit; each one solely responsible for the authentication of its users. In general this meant that the system administrator would have to create an account for each user for each service, and the user would have to authenticate for any given service each time it was used by him or her.

This also meant that each machine on a network had to be updated every time a new user was added if that user was allowed to log-in. The original synchronization mechanism (at least in the UNIX world) was a simple copy of the passwd file. As networks grew larger, this became such a headache that eventually the concept of a directory server was conceived. In simple terms, a directory server is a centralized database that holds hierarchal data, such as users, groups, or anything else.

LDAP was the most prolific example of a directory server and is still used today. Although it solves the problem of duplication, there is still a minor annoyance encountered by today’s users of having to authenticate each and every time a service is used. Ideally we would want for a user to sign in, have secured access to all of the services offered by the network, and be pre-authenticated as the correct user. A naive administrator might find the concept easy to implement, but if one seeks a secure and reliable implementation, it is indeed a difficult problem. Today this idea has come to be known as single sign-on, and it is the topic of our group’s research. Single sign-on is a technique used to validate the user's identity only once and give him access to all of the network's services without having to constantly prompt the user for his or her password.

We first looked at the motivation for single sign on. Aside from the obvious benefit of being more convenient for the user, it is also makes life (eventually) much easier for a system administrator. The system administrator would only have to worry about one single core group of users instead of various users for each service. Since users only have to worry about one set of credentials, it makes things easier for both parties (imagine helping a user who forgot their password for every service on the network) [11].

The most popular mechanism used to achieve single sign-on is Kerberos, which is what our group has chosen to try to set up. We hope to learn what kinds of challenges prevent such an innovative solution from being the norm on today’s networks. We also hope to gain experience in setting up various services (such as SSH, FTP, NFS, and Mail) to work with Kerberos. Our ultimate goal is to build a small network (2 or 3 computers only) that accomplishes single sign-on using LDAP+SSL and Kerberos as well as several services upon which we can test the single sign-on.

Our group’s final turn-in includes 3 working virtual machines on a network that will allow us to perform SSH, mail, FTP, and NFS while only prompting for the user’s password once.

2. Analysis

LDAP

LDAP is a Lightweight Directory Access Protocol. Using LDAP we can add, modify, and query for information. Conventionally it is only used for users and groups, but it can also be expanded to hold information about DNS, or any other service that can utilize a database [9]. LDAP is hierarchal by nature. The root node in the hierarchy is usually the domain for which the server will be responsible (for example, uccs.edu). From there it normally branches off into organizational units, which is then followed by user information.

For our project, we decided to use OpenLDAP, which is an open source project. OpenLDAP was created in 1998 and is loosely based on the LDAP server at the University of Michigan (presumably an in-house program) [10]. The OpenLDAP server has the ability to clone itself to other locations for increased robustness, but we decided not to do any replication for our project. Normally, a stock install of OpenLDAP uses an insecure communications mechanism. Given the motivation for this project, we decided to enable the SSL capabilities of OpenLDAP.

Other choices of LDAP servers included:

- Active Directory by Microsoft
- Open Directory by Apple
- eDirectory by Novell
- Red Hat Directory Server by Red Hat

We chose OpenLDAP because (1) it is open source (and therefore, free), and (2) we wanted a challenge.

SSL

The Secure Socket Layer is a protocol that is used to ensure that data transferred over networks are encrypted. This helps prevent attackers from tampering with or eavesdropping on the data [12].

We, again, decided to use another open source project called OpenSSL, which implements both SSL and the newer protocol, TLS (Transport Layer Security). The choice was based mostly on

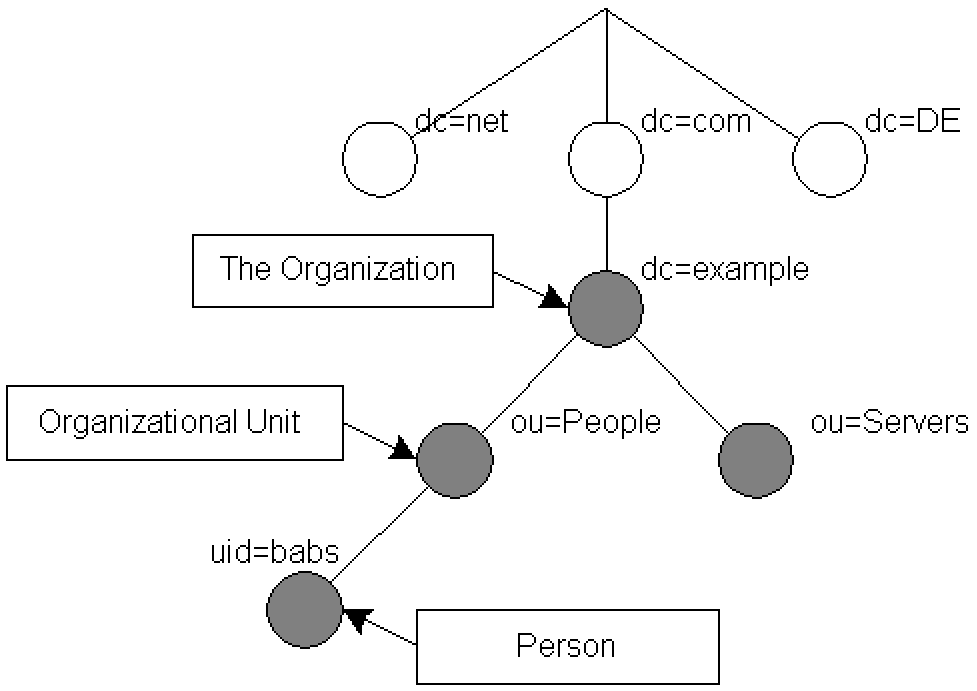


Figure 1: LDAP Directory Structure

convenience because it came bundled with our Linux installs.

Kerberos
Kerberos is both a protocol and an application. The protocol describes a way to securely prove one's identity over a network. The program is an open source implementation of the protocol and was developed by the Massachusetts Institute of Technology (MIT). Using Kerberos allows single sign-on to occur, so it was a natural fit for our problem.

Kerberos has two parts: an authentication server and a ticket granting server [5]. Normally when a user needs to do something for the first time (usually in our context this means logging on to a computer) the user's identity can not be established, so the user must prove his or her identity by authenticating against the authentication server. This is, so far, exactly the same as an LDAP approach.

Once the user has successfully authenticated using a password or some other mechanism, the protocol then invokes the ticket granting server. As the name implies, this server grants tickets. A ticket is simply a piece of data that lets the user prove his or her identity in the future. Under the hood, these tickets use symmetric key cryptography in order to accomplish this. The tickets expire after a period of time, but as long as the user stays signed in they will continue to be re-issued a ticket, so in actuality, they will never have to type in their password more than once.

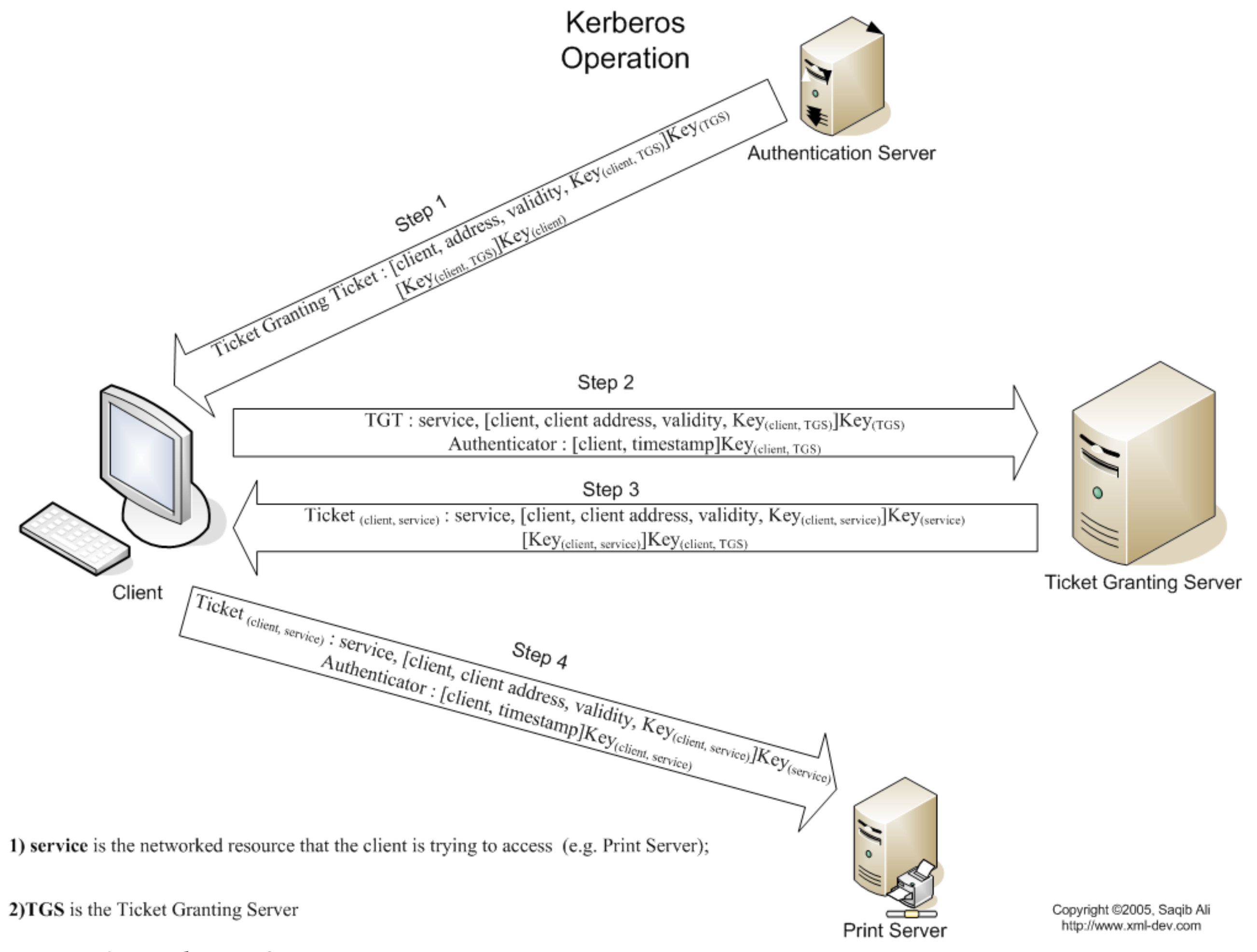

Kerberos Operation
Authentication Server
Step 1
Ticket Granting Ticket : [client, address, validity, Key$_{(client, TGS)}$]Key$_{(TGS)}$
[Key$_{(client, TGS)}$]Key$_{(client)}$
Step 2
TGT : service, [client, client address, validity, Key$_{(client, TGS)}$]Key$_{(TGS)}$
Authenticator : [client, timestamp]Key$_{(client, TGS)}$
Step 3
Ticket $_{(client, service)}$: service, [client, client address, validity, Key$_{(client, service)}$]Key$_{(service)}$
[Key$_{(client, service)}$]Key$_{(client, TGS)}$
Client
Ticket Granting Server
Step 4
Ticket $_{(client, service)}$: service, [client, client address, validity, Key$_{(client, service)}$]Key$_{(service)}$
Authenticator : [client, timestamp]Key$_{(client, service)}$
1) **service** is the networked resource that the client is trying to access (e.g. Print Server);
2)**TGS** is the Ticket Granting Server
Print Server
Copyright ©2005, Saqib Ali
http://www.xml-dev.com

Figure 2: Kerberos Operation

If the user wants to use a service on the network using single sign on, there are two prerequisites. First, the user must already have their ticket. Second, the service must explicitly support Kerberos authentication; otherwise, the user will still need to authenticate the traditional way (whatever that happens to mean for the service in question). The user gives his or her ticket to the service. Using the user's ticket, the service transparently authenticates the user and allows or denies access, all without ever prompting the user for a password. Exactly how this accomplished is rather complicated but suffice it to say that when it works, it works seamlessly.

On larger corporate networks, the authentication server and ticket granting server are separate, but because our network is small, they are both on the same server.

One last thing to note about Kerberos is that time is a very important parameter in the protocol. As such, both the client and server must have synchronized clocks, so the usage of NTP (network time protocol) is essential.

3. Design

Our setup consists of three virtual machines that are named Cartman, Stan, and Kenny. All use Debian variants for consistency's sake.

Cartman is the central server and contains the following:

- Debian Lenny
- LDAP
- Kerberos
- NTP server
- SSH server

Stan is the secondary (non-infrastructure) server and contains and is responsible for the following services:

- Debian Lenny
- SSH server
- Mail
- NFS
- FTP

Kenny is our client machine and contains the following:

- Ubuntu 8.04
- SSH server

Both Kenny and Cartman are mounting Stan's NFS share (which is the /home directory) so that users will have the same home folders on any of the machines. To keep ourselves honest, we do NOT accept RSA or DSA keys in SSH (we could make it look like single sign-on when actually it is not), nor is the mail client on Kenny supposed to store any passwords.

4. Implementation

LDAP

The first thing we had to do was get LDAP up and running. Although Kerberos will eventually authorize the logins, LDAP still serves as a base for all other user information, such as user id, groups, home directory, login shell, full name, etc. As previously mentioned we chose Cartman for this task. We used a BDB database for the back end and had very few issues when it came to getting basic LDAP working.

The only challenge we faced was figuring out the slight differences between Debian and Ubuntu in terms of client configuration (they put their configurations in different files). Once we figured this out, it was smooth sailing.

We had to tell the name service to use LDAP (the name service simply maps user ids to names and vice-versa). Then we had to configure PAM (Pluggable Authentication Modules) to authenticate against LDAP. Once this was done, we removed all of our local accounts on all of the machines, and everything was working.

A quick word about caching: Linux has a nasty habit of caching login credentials, so we learned it is important to turn off caching while experimenting with a new setup. Otherwise, you will be scratching your head until trying to figure out why some things work and others do not.

SSL

Adding SSL to the LDAP was a bit more difficult. The first thing we had to do was generate the certificates like we did for the class in Homework 4. Essentially all that SSL provides is some privacy. No peer verification is done per se, but the clients can be sure that their traffic to the server will not be seen by anyone sniffing the traffic.

We ran into problems with the certificates because a couple of tutorials stated that we should become our own Certificate Authority and validate Cartman with ourselves. This did not work out because we had a few errors with pointing to the right certificates. However, we were able to debug those fairly quickly once we had the configuration files correct.

Another problem we encountered was one of nomenclature. In various places we would see references to ldaps (LDAP over SSL), and others would mention StartTLS. No one was entirely clear on the difference, but we found out that it was a matter of protocol versions. LDAP version 2 did not originally support SSL at all, so they decided to run an SSL-only server on a different port instead (ldaps, or port 636). The newer version 3 supports either plain or SSL on the same port and uses the StartTLS mechanism to accomplish this.

By this point everything was using ldaps, so we simply changed back to ldap and enabled StartTLS, and our woes came to an end. For added security, we made it so that the clients had to have a local copy of the server's public key (in other words, the server will not give out its public key to anyone). This is only a stopgap measure, but what the heck.

Kerberos

Installing Kerberos on both the server and clients was about as simple as it can get. Configuring it on the other hand was a bit more involved. The first step involved creating a realm. A realm is the Kerberos equivalent of a domain. Each realm needs an authentication server and a ticket granting server. Realms can service more than one domain, but for our case it will only service vast.uccs.edu, so the realm is VAST.UCCS.EDU (Kerberos likes upper case for realms).

Once the configuration files were told about our realm, we had to create principles for the realm. Principles are really anything that might ever need authenticating. They can be users but can also be machines (Cartman, Stan, and Kenny all have their own principle), services, or Kerberos administrators.

The hardest part is creating the first user. Normally Kerberos requires an authenticated user in order to be administered, but since there aren't any to start with, we had to bootstrap Kerberos by temporarily disabling this requirement. This lets anyone (literally anyone) administer Kerberos. Once there, we added a principle for the administrator (which includes a password), and then re-enabled the security on Kerberos.

At this point we started wondering how to integrate this into our existing LDAP setup, but our research yielded that we had misunderstood Kerberos. As it turns out, Kerberos completely replaces the authentication portion of LDAP, and for a number of pedantic reasons, it cannot use LDAP directly. So we had to migrate (in other words, manually create a principle for each of) the users. At this point it became clear why some people may not want to use Kerberos. As marvelous an idea as it is, it would be a real headache to migrate an existing LDAP infrastructure over to Kerberos. The real kicker is that the users must re-type their passwords for Kerberos (its hashing mechanism is different). Thus, any larger networks would simply be unable to employ the use of Kerberos.

Once the users were set up, we had to make sure that the machines would log them in using Kerberos instead of LDAP. This was a one-line change in the PAM configuration files, but the Kerberos clients had to have their keystores configured (a keystore simply stores certain passwords for a host).

At this point a user can log in to any machine and be allowed to log in. In order for single sign-on to work, each service must now be configured in order to support Kerberos. And the first one we chose was SSH since it would make testing everything else much easier.

SSH
This was one of the easier services to configure. First, we had to install OpenSSH. Luckily, OpenSSH supports Kerberos out of the box with a few quick tweaks. Mainly we had to enable GSSAPI (a mechanism for authenticating against Kerberos) in the server and had to tell the SSH client to pass its GSSAPI credentials along to other servers.

Once this was done on all three of the machines, we could finally see the beginnings of our single sign-on network taking shape. We could SSH to any of the three machines and at first be prompted for our password. From that point on, any further SSH sessions within the realm were completely password-less (again, without resorting to using OpenSSH's built-in PKI). We ran into a few issues, mostly related to the aforementioned keystores. Once we had made sure that the right hosts had the right keys, everything worked perfectly.

Mail
By far this was the most difficult service to install. We chose this one because it epitomizes the motivation behind this project. What could be more impressive than logging on and having password-less access to your email? It took quite a bit of work, but we did manage to get it working.

At first we chose Postfix for the SMTP server and Courier for the IMAP server, but after an agonizing search, we finally learned that Courier does not support Kerberos. Our next choice was Dovecot, which claimed full support for Kerberos. We thought it would then be easy but learned later how wrong we were.

Before configuring Dovecot we decided to tackle SMTP. Ideally our goal was to have both IMAP and SMTP authenticate using Kerberos credentials, but after a long frustrating time, we found out that Postfix was just a horrible nightmare to setup with Kerberos (involving other go-between services). Ultimately, it was decided that it did not matter since Postfix was only the MTA, and from the fictional client's point of view, all they wanted to do was receive mail (not in a real setup of course). Thus, we configured Postfix as a very simple MTA to deliver our mail to a user's home directory.

Then we had to set up Dovecot. Obviously, the first thing we did was switch it from standard IMAP to the more secure IMAPS (a one-liner). Then we had to hook it in to Kerberos. We told it to authenticate against Kerberos, which involved creating a separate keystore for some reason as well as describing the realm to the mail server.

We also had to make it use LDAP for any user information (such as where the user's mail was located), which is where we encountered another problem. It turns out that Dovecot really cannot handle our StartTLS LDAP setup, so we had to search for a different route. An article on-line suggested that we have it use PAM instead since PAM is using LDAP as its back-end. It was a roundabout way of doing the same thing, but it worked.

At this point we thought we were done, so we logged into Kenny's desktop, fired up Thunderbird, and tried to connect to the server. Although the SSL and initial communication worked, the authentication failed miserably. After parsing through the cryptic logs, we found that Dovecot was trying to use a principle that did not exist ("imap/stan"). It was at this point that we learned that some services required a separate service principle for their respective hosts.

After a lot more fudging around, we finally got mail working. To make sure it was really working we turned off plain (non-Kerberos) authentication from within Dovecot, and amazingly it still worked.

FTP

The FTP setup might have been the easiest one to get working. We setup the FTP server on Stan, which is the secondary server in charge of services. The needed package to setup FTP was called krb5-ftpd, which was a "Kerberized" version of FTP.

The only challenge that we faced in the setup was not realizing the daemon, inetd, was not installed. Once that was found adding an FTP listener line in the "inetd.conf" file was all that was necessary for the server to be operational.

Kenny was used to test FTP. Rather than call FTP, the command krb-ftp was used because it is the Kerberos version found in the krb-client packages. This confirmed that single sign-on FTP was working.

NFS

Traditionally, NFS is not a very secure protocol. It is used for file sharing. Other machines can mount an NFS share. The only security offered is that a server can allow only certain IP addresses to connect. This is fine in a closed network , but there are some situations where this does not work. The newest version of NFS (NFS4) supports Kerberos authentication to address this concern. We decided to try and set this up as our last service.

Stan was the NFS server (no particular reason other than we decided that Cartman would only be responsible for LDAP and Kerberos). After a bit of research, we found that none of this would work unless we drastically changed our keystores. Normally a keystore keeps a password in a number of different hash forms, but NFS4 only likes one. Thus we had to re-do the keystores on all 3 machines, which was a huge pain. In retrospect it would have been better to do this service first.

Once the keystores were correctly set up, we had to create an export on Stan. We agreed that the home directories were a logical choice. One slight difference that we found between NFS3 and NFS4 is that version 3 allows you to export anything, and mount it on the client with the same name (in other words, "/home" on the server is "server:/home" on the client). On NFS4 this is not the case ("/home" on the server maps to "server:/" on the client). This is just a security feature.

At this point we thought we were finished, but every time we tried to mount the server the client would simply freeze. It turns out that the server had to be configured to run a few additional NFS4-specific programs. After this we tried again only to find that the clients also had some tweaks required. We had no problems after this, and the client mounted the server just fine.

One last thing we decided to do (not required, just nice) was to make NFS use autofs. In essence autofs does two things: keeps the load on the server down and remounts a share if it is lost for some reason. So if the server goes down for some reason autofs will fail quietly rather than freezing up the client machines. At this point we called it good.

5. Performance Evaluation

We did not plan on using any metrics for our project because there really wasn't anything that had to be measured. Our main goal for our project was to get single sign-on working with the various services we setup on the network.

After the the initial log-in and password authentication, we are able to do the following:

1. We can SSH between Cartman, Stan, and Kenny without every being prompted for a password except for when we first log-in.
2. We can use FTP from Kenny to view and transfer any files we want to and from Stan without being prompted for a password.
3. We can mount server directories on Stan from Kenny without being asked for a password.
4. We can send and receive mail using Thunderbird on Kenny without ever being prompted for a password.

Single sign-on with Kerberos worked as expected with the network services.

6. Future Directions

There are a few things we can do to help improve our project such as the following:

- Add a network firewall
- Add more single sign-on services such as Apache
- Add multi-platform capabilities such as Windows
- Add security to SMTP

7. Conclusion

Our project involved getting three virtual machines running with a variety of services in order to setup single sign-on using Kerberos. While using a lot of time and a little bit of frustration, we were able to get single sign-on to work with the different services we setup. Any user who logs into our network should only be prompted for his or her password only once and should have authorized access to any network service. In a nutshell, our project was a success.

8. References

[1] Debian, Dec. 2008, http://www.debian.org/.

[2] Dovecot, Nov. 2008, http://www.dovecot.org/.

[3] Gentoo Linux Wiki, Nov. 2008 http://en.gentoo-wiki.com/wiki/Main_Page.

[4] Kerberos: The Network Authentication Protocol, Oct. 2007, http://web.mit.edu/kerberos/.

[5] OpenLDAP, March 2008, http://www.openldap.org/.

[6] OpenSSL, Nov. 2008, http://www.openssl.org/.

[7] Ubuntu Forums, http://ubuntuforums.org/.

[8] Wikipedia, “Kerberos (protocol)”, Nov. 2008, http://en.wikipedia.org/wiki/Kerberos_(protocol).

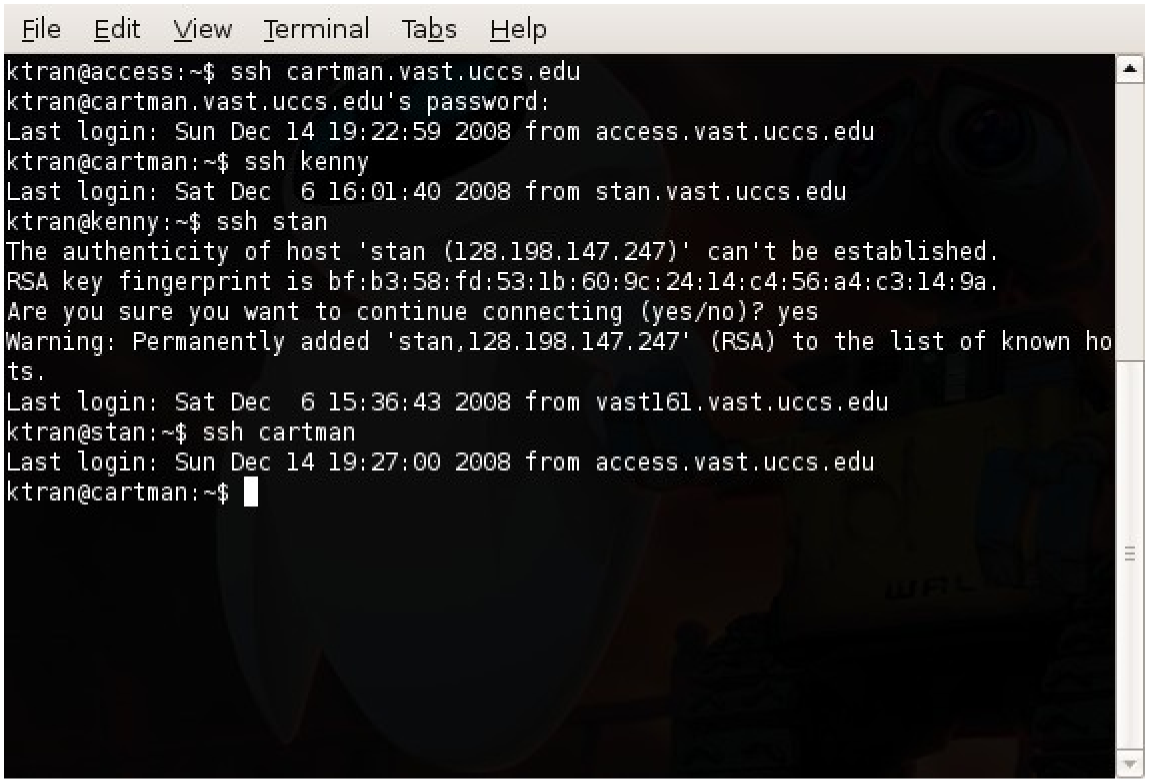
[9] Wikipedia, “Lightweight Directory Access Protocol”, Dec. 2008, http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol.

[10] Wikipedia, “OpenLDAP”, Nov. 2008, http://en.wikipedia.org/wiki/OpenLDAP.

[11] Wikipedia, “Single sign-on”, Dec. 2008, http://en.wikipedia.org/wiki/Single_sign-on.

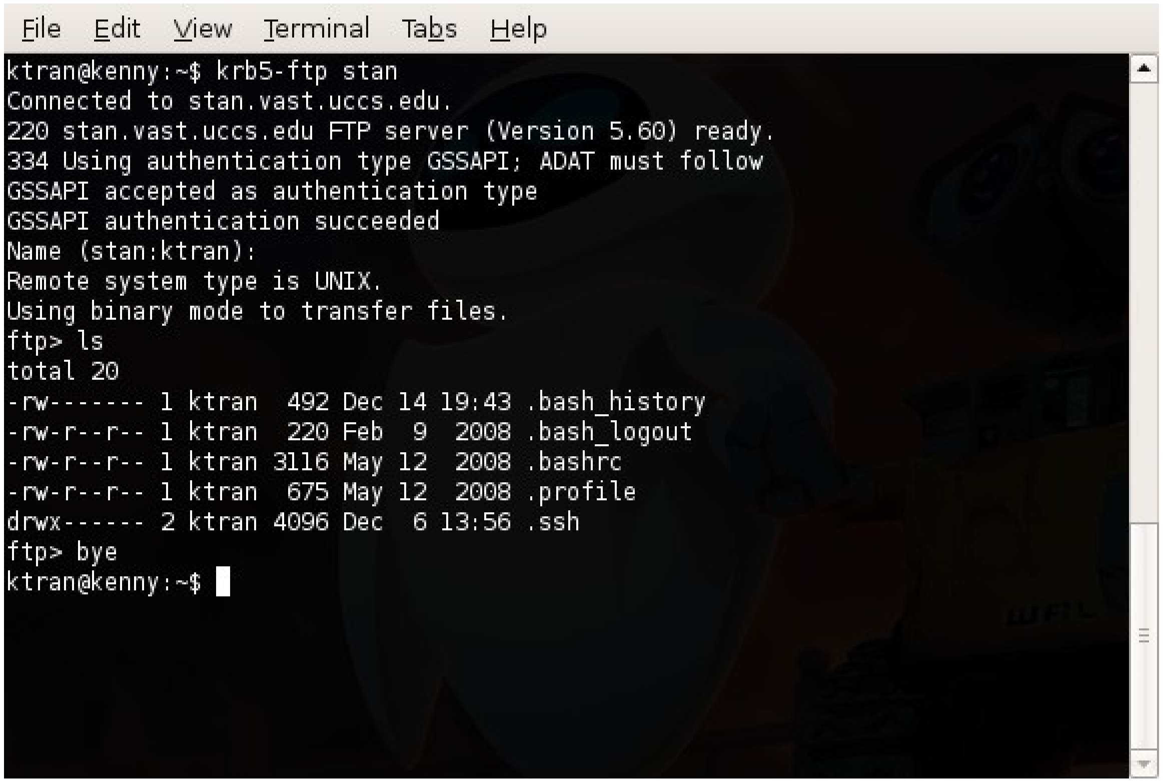
[12] Wikipedia, “Transport Layer Security”, Dec. 2008, http://en.wikipedia.org/wiki/Transport_Layer_Security.

Appendix A



```
ktran@access:~$ ssh cartman.vast.uccs.edu
ktran@cartman.vast.uccs.edu's password:
Last login: Sun Dec 14 19:22:59 2008 from access.vast.uccs.edu
ktran@cartman:~$ ssh kenny
Last login: Sat Dec  6 16:01:40 2008 from stan.vast.uccs.edu
ktran@kenny:~$ ssh stan
The authenticity of host 'stan (128.198.147.247)' can't be established.
RSA key fingerprint is bf:b3:58:fd:53:1b:60:9c:24:14:c4:56:a4:c3:14:9a.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'stan,128.198.147.247' (RSA) to the list of known ho
ts.
Last login: Sat Dec  6 15:36:43 2008 from vast161.vast.uccs.edu
ktran@stan:~$ ssh cartman
Last login: Sun Dec 14 19:27:00 2008 from access.vast.uccs.edu
ktran@cartman:~$
```

Figure 3: Single Sign-on SSH Demonstration



```
ktran@kenny:~$ krb5-ftp stan
Connected to stan.vast.uccs.edu.
220 stan.vast.uccs.edu FTP server (Version 5.60) ready.
334 Using authentication type GSSAPI; ADAT must follow
GSSAPI accepted as authentication type
GSSAPI authentication succeeded
Name (stan:ktran):
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
total 20
-rw------- 1 ktran  492 Dec 14 19:43 .bash_history
-rw-r--r-- 1 ktran  220 Feb  9  2008 .bash_logout
-rw-r--r-- 1 ktran 3116 May 12  2008 .bashrc
-rw-r--r-- 1 ktran  675 May 12  2008 .profile
drwx------ 2 ktran 4096 Dec  6 13:56 .ssh
ftp> bye
ktran@kenny:~$
```

Figure 4: Single Sign-on FTP Demonstration