Kerberos transition started

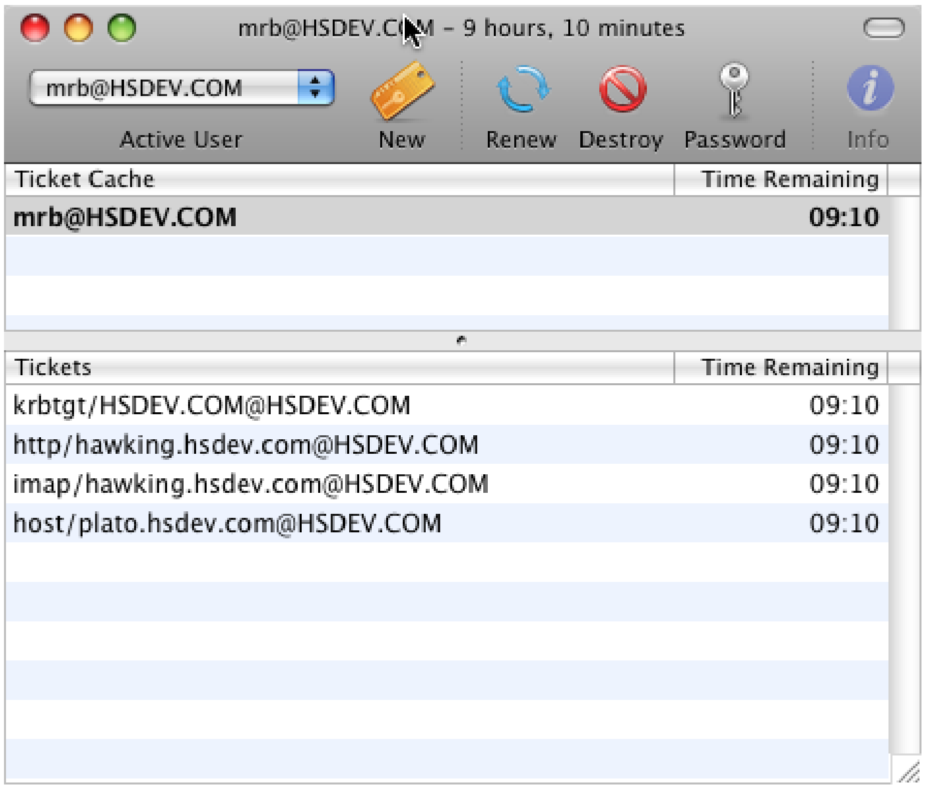
As of last thursday, I've started to migrate everything in our network to use GSSAPI or Kerberos authentication. The amount of passwords and accounts grew over our heads and the inevitable "I'll use the same everywhere" started to be apparent.

The actual preparation for this already started more than a year ago. We are in a mixed environment of several Windows varieties, Linux servers and a couple of Macs. It's a fairly small network, but complex enough to easily make mistakes or forget something.

The installation of the Kerberos admin server and the domain controller is fairly straightforward if you play by its strict rules. Especially shortcuts in terms of how hostnames get resolved to ip-addresses and vice versa has very little playing room. I had to adapt every `/etc/hosts` file on every machine to get it to work.

Another thing which Kerberos really does not like is ip-addresses behind a NAT router, be they private ip-addresses or not. We got the actual authentication to work properly across routers, but the password changing only works from machines on the same subnet as the Kerberos server itself. Perhaps I should replicate a slave on each subnet and do password changes against them? (Not sure if that is possible though)

On the client side, I work on OSX myself so that was the first place to convert. Support for Kerberos is built into OSX, although the ticket manager is sort of hidden.



Ticket Cache	Time Remaining
mrb@HSDEV.COM	**09:10**

Tickets	Time Remaining
krbtgt/HSDEV.COM@HSDEV.COM	09:10
http/hawking.hsdev.com@HSDEV.COM	09:10
imap/hawking.hsdev.com@HSDEV.COM	09:10
host/plato.hsdev.com@HSDEV.COM	09:10

The way Kerberos works is sort of like an attraction park. You **pay** for the entrance ticket by entering your username/password; shown in the upper part of the window above and you get free tickets to all the attractions in the park; shown in the lower part of the picture. In this case there is the Ticket Granting Ticket (the person who hands out tickets if you will), a ticket for the 'http' service on hawking.hsdev.com (an iCal server in fact), the 'imap' service on hawking and a host service (ssh) on plato. While doing all this, traffic is encrypted and your password never travels over the network, so that gives most system administrators a good feeling. The traffic of the service itself is not affected by Kerberos, it just does the authentication, nothing more.

When opening a service like imap, http or whatever has been registered with the Kerberos Realm, there's no need to enter a password again, after the initial ticket has been granted within the realm. Even in a small network, this can save a big amount of time.

Ok, after the Kerberos installation, with the 'host' services as the initial service to enable on all hosts, the real work started. I made a (big) list of everything in our network which in some way asks for a password. Mail and Calendaring came out on top, not only because everyone used this, but also because I knew my clients were easy to configure for it, so I could focus on the server configuration.

Surprisingly, because the iCal server runs on Linux, for which it has not been written primarily, was the easiest. Changing the accounts.xml to have empty passwords for the users and putting in the Kerberos realm in the configuration file was enough to get it working.

For mail, there are three services involved, imap and smtp and sieve. Cyrus uses the sasl library for authentication and we used its internal database for authentication. Making cyrus on debian use GSSAPI is easy, I dont think I had to change anything in fact. The problem is in the multiple ways people tend to get to their mail. Next to their desktop clients there are webmail accesses, iPhones and specific uses of some shared mailboxes. All of these should ideally support Kerberos, but they dont. Until they all do or have been made to do so, the internal sasl database will need to stay for a couple of accounts and thus multiple passwords will still float around.

I found that most uses can be made to use Kerberos if needed, be it direct or indirect. Many web applications have the option to use http authentication, which in turn can use an apache module to ask Kerberos for user and password information. It's not ideal, because control over http authentication sucks (hard to logout, for example), but it will have to do if the goal is to minimize the amount of passwords.

The amount of steps ahead is still massive, but the first steps are there. The list of things to do also made me look at services more critically and review if we really needed them to be there. I'm guessing the completion of all the tiny little bits will take the best part of the rest of the year.

Next up is ejabberd, our clients are iChat and Spark, which both can be Kerberos enabled, so the client side should be easy.