Linux Authentication: Part 2

THE KERBEROS NETWORK AUTHENTICATION SYSTEM

In the second article in this series Bruce Richardson looks at how Kerberos can be used to implement a centralised network authentication system

What is Kerberos?

Kerberos is an authentication system designed to provide secure remote authentication and encrypted access to network services based upon that authentication. It's fast, relatively easy to set-up, an open standard and Open Source software (though proprietary implementations of the standard also exist).

Kerberos was developed at MIT as part of Project Athena, the university's distributed network computing project. Kerberos is also the name of the three-headed dog which, according to Ancient Greek legend, guards the entrance to the underworld. Due to its strong encryption, the full MIT Kerberos code is classed as a munition and cannot be obtained outside the US. To get around this, a version of the code was stripped of all the encryption (and given the slightly ghoulish nickname "E-bones". Developers at the Royal Institute of Technology in Stockholm then re-implemented the encryption. Their version of the code is called "Heimdal", named after the Viking god who guards the entrance to Valhalla.

The Kerberos protocol is currently on version 5. Version 4 was the first version that was stable and secure enough for practical use but has significant disadvantages compared to V5 and should be treated as an item of historical interest only.

How does it work?

The Kerberos authentication system is based on tickets. It involves a simple 3-step process:

- You identify yourself to a service.
- The service grants you a ticket.
- You use that ticket to get access to network resources.

The first time you go through this process is when you login to your Kerberos realm. In step one you identify yourself to the Key Distribution Centre (KDC) by giving it your password. The KDC grants you an initial ticket. This ticket will act as proof of your identity until it expires (eight hours is the default lifetime).

When you access a Kerberos-aware service (see the section called Kerberos-ready applications) you go through this process again. In step one you identify yourself to the service by showing it your initial ticket. The service checks your ticket with the KDC and then gives you another ticket, which enables you to access its resources. That ticket is usually good for one session (login session, mail retrieval or whatever). Due to the initial ticket's role in getting you further tickets it is usually referred to as a Ticket Granting Ticket (TGT).

The Kerberos network model

To understand how Kerberos works you should be familiar with the key components of a Kerberos network.

The Realm is the organisational unit of the Kerberos network, comparable in may ways to the NT domain. Each Realm is associated with a KDC and Admin. server. It is entirely up to the system administrator how realms are named and which users/machines/services are members of which realms. The convention, however, is to map Kerberos realms to DNS domains and to give the realm the same name as the corresponding domain, only upper case (realm names are case-sensitive). So the realm for charity.org would be CHARITY.ORG.

If no domain is specified, as a command-line argument or in a config file, Kerberos software will assume that this convention has been followed. It is possible to establish trust relationships between realms, so that users on one realm may access services on another. This article does not go into that.

Each realm has at least one Key Distribution Centre, which stores the password database and grants Ticket Granting Tickets. If a realm has more than one then one is the master and the others are slaves, synchronising their databases from the master.

It is essential to keep your KDC secure: if it is cracked then your whole network is compromised. The administration server allows the Kerberos database to be manipulated remotely, enabling an administrator to add accounts, change passwords

etc. It is not essential to run one: you could make all changes while directly logged in to the KDC, which would be secure if limiting. Admin servers are usually run on the same host as the KDC for convenience and security but this is not a requirement.

Tickets

Every service available through Kerberos requires a ticket. Each service requires a different kind of ticket but all tickets have these things in common:

- They are issued to a specific principal, granting access to a specific service.
- They have a fixed lifetime, after which they expire if not explicitly renewed.
- They are issued for a specific host. That is, by default they can only be used from the host on which they were requested (see the section called A typical user session).

Credentials cache

The Credentials cache stores all the tickets you have been issued during your current Kerberos session. By default this is a file in /tmp readable only by you, but this is configurable. It is possible to open multiple concurrent Kerberos sessions, in which case you will have multiple caches.

A typical user session

Fred is already logged on locally at his Linux workstation but hasn't yet logged in to the Kerberos realm. To do this he uses the kinit utility:

```
$ kinit
fred@CHARITY.ORG's password:
```

Because the local kerberos config files do not specify a domain and because he passed no special arguments to kinit, kinit assumes that his principal name matches his local account name and that the realm is the upper-case version of the local DNS domain. Luckily, this is correct and once he has typed in his password he is issued a TGT.

The hosts on Fred's network run kerberised telnet daemons. Fred decides to log into his network's mailhost:

```
$ telnet -x -l fred mailhub.charity.org
trying 192.168.10.12...
Connected to mailhub.charity.org
(192.168.10.12)
Escape character is '^]'
Negotiating encryption...
Last Login: Dec 22 14:03:45 from
workstation.charity.org
$
```

Note that Fred didn't need a password and that his Telnet session is encrypted.

Principals

A Kerberos principal is roughly analogous to a Unix account. It may represent a human user, a machine or a network service. Principal names are constructed from up to three components (in practice you will never see more than two) and the realm name, in the form component/component/component@realm. The first component is referred to as the name, the second as the instance and there is as yet no standard use for the third.

A typical principal name would be fred@CHARITY.ORG. Just as there is a convention to map realm names to DNS domains, so there is one to name user principals after Unix accounts. As a result, user's principal names often match their e-mail addresses.

If Fred were an administrator then he would usually also have an account fred/admin@CHARITY.ORG, which he would use to access the admin server. Note that although this extra account is referred to as Fred's "admin instance" there is in fact absolutely no link between the fred@CHARITY.ORG and fred/admin@CHARITY.ORG. They are completely separate principals with different passwords and network privileges. Fred could log into the admin server as vendingmachine/repairman@CHARITY.ORG if there were such an account. It is simply the convention to name organisationally related accounts in this way. If Fred runs the kadmin utility without specifying a principal then it will assume that fred/admin@CHARITY.ORG is the principal as whom it should try and connect.

At this point, Fred remembers that he has something he needs to do on the proxy server. So:

```
$ telnet -x -l fred squid.charity.org
trying 192.168.10.1...
Connected to squid.charity.org (192.168.10.1)
Escape character is '^]'

Debian GNU/Linux 3.0 squid

squid login:
```

Oops. Fred forgot that Kerberos tickets are, by default, only good for one host. His TGT is no good on mailhub – in fact they don't even exist on mailhub, having been left behind on workstation. So the kerberised telnet service fell back on the plain old unencrypted and insecure standard.

Now, Fred could run kinit on mailhub but that would be insecure: the whole point of kerberos is

that your password is not transmitted across the network. So he logs out of mailhub, returning to workstation. Out of curiosity he checks to see the details of the tickets he has acquired so far:

```
$ klist
Ticket file:         /tmp/krb5cc_1002
Principal:           fred@CHARITY.ORG
Issued               Expires        Principal
Jan 05 12:37:22  Jan 05 20:37:22
krbtgt/CHARITY.ORG@CHARITY.ORG
Jan 05 12:38:12  Jan 05 20:37:22
host/mailhub.charity.org@CHARITY.ORG
```

This shows him the original TGT and the Telnet ticket from mailhub. Note that the Telnet ticket expires at the same time as the TGT used to obtain it: a service ticket may expire before the original TGT but may not outlive it.

But Fred wants to start afresh, so

```
$ kdestroy
Tickets destroyed
$ kinit -f
fred@CHARITY.ORG's password:
```

His new ticket is now forwardable. If he re-runs Telnet, adding an -F option, his TGT will follow him to the new host and to any other host he telnets into from there. If he runs telnet with the -f option then his TGT will follow him to the new host but will not be further forwardable (i.e. if he telnets to mailhub without a password from there to squid but not from squid to anywhere else.)

For a glimpse under the hood of Kerberos, have a look at the sidebar A Kerberised Telnet session in detail, which gives a detailed technical account of how the telnet session is authorised. One thing to take particular notice of is the paranoid and secure fashion in which Kerberos creates an encryption key for the session. It is this key which provides the mechanism for encrypting the subsequent telnet communications. In this fashion any properly kerberised application can enjoy the benefits of secure encrypted operation across the network.

Using Kerberos on your network

Unless you are a highly skilled developer, there are essentially two ways to use Kerberos on your network:

- Install services (and clients to access them) which have already been developed to use Kerberos. Do check the documentation to see how fully the application supports/uses Kerberos: some applications only use it for authentication, others make full use of its features to enable secure, encrypted communication.
- Install services/clients which use a generic high-security mechanism (e.g. SASL, GSS-API) that can use Kerberos as a backend. These generic security layers are actually more complex than Kerberos and an application that properly supports them can make full use of Kerberos security.

A Kerberised Telnet session in detail

To give an idea of how paranoid Kerberos security is, here is that Telnet session in detail:

- Fred sends a request (using his Ticket Granting Ticket) to the KDC: "I want to talk to the Telnet daemon on charity.org" (well, the kerberised Telnet client does it but let's keep this simple).
- The KDC generates a new session key, which Fred and the Telnet daemon will use to secure their communication.
- The KDC sends two messages to Fred: the first contains a copy of the new key and the name of the remote Telnet daemon and is encrypted using Fred's key. The second contains a copy of the new key and Fred's name and is encrypted using the Telnet daemon's key (and is Fred's "ticket" to talk to the Telnet daemon).

Note: The KDC is not involved from this point on.

- Fred decrypts the first message (he can't decrypt the second as he doesn't have the key) and extracts the new session key.
- Fred creates a message containing the current time (the "authenticator") and encrypts it using the session key.
- Fred sends the new message and the ticket he received from the KDC to the Telnet daemon.
- The Telnet daemon decrypts the ticket from the KDC (passed on to it by Fred) and extracts the session key and Fred's name.
- The Telnet daemon uses the session key to decrypt the authenticator from Fred and checks the time.
- At this point, Fred has authenticated himself to the Telnet daemon and they can use the session key for further communication. But Fred may want the Telnet daemon to authenticate itself to him, in which case:
- The Telnet daemon takes the timestamp from Fred's authenticator, adds its name and encrypts the result with the session key to create its own authenticator, which it sends back to Fred.

- Install the PAM Kerberos 5 module and use that to integrate Kerberos into your network authentication policy.

Kerberos-ready applications

The Kerberos source comes with a selection of kerberised replacements of standard Unix apps (Telnet, ftp, rsh etc). While these are interesting to experiment with they are based on creaky old code and I wouldn't advise using them seriously on your network. Kerberised versions of the more recent Linux apps are out there.

There is an ever-increasing number of serious applications available using Kerberos authentication, either directly or through GSS-API or SASL. This includes PostgreSQL, OpenLDAP and Cyrus IMAP. Of particular interest is Cyrus IMAP, which will not only use Kerberos for authentication and encryption but can also use it to store group membership information (Cyrus employs a sophisticated system of group and user permissions to allow access to mail folders). Of course, you'll need a mail client that can use these security mechanisms. Mutt is a good example for Unix and the respected Eudora mail client does the same for Windows.

One very interesting Kerberos-based application is the Andrew File System, which uses the Kerberos security model to provide a distributed network filesystem. It's rather more sophisticated than NIS and much more secure!

PAM

PAM offers the crudest way to integrate Kerberos into your network. PAM offers a relatively simple authentication interface with no provision for the encrypted communications features of Kerberos. Still, if you add the kerberos module to the stack of the Linux login app then it will authenticate the login against the KDC, fetch a TGT, store it and destroy it when you logout. If you combine the kerberos module with the mkhomedir module, which automates the creation of local home directories for newly authenticated users, you can implement your own roaming logon system (assuming you are fortunate enough to have Linux desktops in your workplace).

Of course, you are all now PAM experts, having read the first article in this series, and will find this no challenge at all.

Working with Windows 2000

You may have heard that Active Directory bases its security model on Kerberos. This is true and although they have, as usual, "embraced and extended" the protocol it is still possible to authenticate users against an Active Directory server and even to create trust relationships between Kerberos and Active Directory domains. See the Info box for details.

An admin session

Fred has to do some admin work on the Kerberos realm. First he needs to connect to the Admin server. Because he doesn't specify a principal, kadmin assumes he wants to connect as fred/admin@CHARITY.ORG. Note that it is only when he asks for Wilma's details that he is asked for his password.

```
$ kadmin
kadmin: getprinc wilma
Principal: wilma@CHARITY.ORG
Expiration date: 2004-01-12 14:22:35
Last password change: 2001-12-22 09:31:05
Password expiration date: 2002-03-22 09:31:05
Last modified: 2001-12-22 09:31:05
(fred/admin@CHARITY.ORG)
Last successful authentication: 2001-12-21 09:35:43
Last failed authentication: 2002-01-05 11:20:19
Failed password attempts: 3
Number of keys: 1
```

If you look at the information Fred retrieved about Wilma, you'll see that she's come back from holiday and forgotten her password. So

```
kadmin: cpw wilma
Enter password for principal "wilma":
Re-enter password for principal "wilma":
Password for "wilma@CHARITY.ORG" changed.
```

That done, Fred wanders off to get a coffee. When he comes back he finds that he has to re-authenticate himself, as the Admin server has been set to grant tickets with five-minute lifetimes to secure it against careless nerds like him. This behaviour differs from that of the kerberised Telnet daemon, which will not abort a telnet session once the ticket expires but will refuse to authorise any fresh ones.

Summary

If this article has done its job then you have learned how Kerberos can bring centralised, secure authentication, user administration and reliable encrypted communications to your network. You've seen practical examples of its use and an overview of its architecture and philosophy. So why aren't you using it? What do you have that's better?

Info

Kerberos FAQ	*http://www.nrl.navy.mil/CCS/people/kenh/kerberos-faq.html*
Heimdal	*http://www.pdc.kth.se/heimdal*
Kerberos for Morons	*http://www.isi.edu/~brian/security/kerberos.html*
Why not use Kerberos?	*http://www.redhat.com/docs/manuals/linux/RHL-7.2-Manual/ref-guide/s1-kerberos-whynot.html*
Win2K Kerberos Guide	*http://www.microsoft.com/windows2000/techinfo/planning/security/kerbsteps.asp*