Understanding Kerberos v5 authentication protocol

Fabrice KAH
GIAC Security Essentials Certification (GSEC)
Practical v1.4b Option #1
November 2003

Abstract

Today more then ever, secure communication is a must. Most companies now use a network infrastructure to conduct their business, whether internally (intranet model) or externally to reach partners or customers (extranet/Internet models).
While it is utopian to consider today's networks as being safe, there are solutions to make them more secure and use them with a bit of trust. The key aspects to securing communications over a distributed environment are authentication, integrity, confidentiality and authorization.
Kerberos is a network protocol that addresses the authentication part. We will discuss in the following document of the principle of Kerberos, its functionalities, but also the integration of this protocol in today's applications.
This guide is intended to give a general knowledge in order to understand Kerberos, and to know what can be done with it; it is not a technical guide.

Introduction

Kerberos gets its name from Greek mythology. Cerberus, also known as Kerberos, was a three headed beast that guarded the Underworld and kept the living from entering the world of the dead. Kerberos protocol design began in the late 1980s at the Massachusetts Institute of Technology (MIT), as part of project Athena. It is a secure authentication mechanism designed for distributed severs, which assumes the network is unsafe. It enables a client and a server to mutually authenticate before establishing a connection. The first public release was Kerberos version 4, which lead to the actual version (v5) in 1993 after a wide public review. It followed the IETF standard process and its specifications are defined in Internet RFC 1510 [1]. Originally designed for UNIX, it is now available for all major operating systems, freely from MIT or also through commercial versions. We will discuss about these later in this document.

Benefits of using Kerberos

What is easier today than to catch credentials over a network? Try to run a sniffer in your environment and you will see. You will certainly get a login/password combination within a few minutes. This could lead to an unauthorized use of your network services and would certainly compromise all data present in your environment; even protected confidential data, as most users are using only one password for every application. Authentication is critical to security. Too many applications use a weak authentication mechanism, like clear text passwords or, even worse, rely on the "honesty" of client applications, known as authentication by assertion: for example Berkeley's "r" services rlogin, rshell, rexec…

However, it is not the primary role of an application to manage security. Consider a mail server: its role is to deliver email messages over the network to the appropriate recipients, but not to verify the user's identity! This is where Kerberos comes in. It has the advantage to manage secure authentication from a central location, and for many applications. For each application that requires this service, it is a reliable, simple and easy to manage solution to use Kerberos. Furthermore, it unloads application servers from this time consuming authentication task and allows concentrating on their primary function.

Kerberos protocol

Encryption

By default, assume information available to anyone else other than the intended recipients is prone to be compromised. It is the case for all data sent over the network, it can be tampered, viewed, modified. Kerberos provides cryptographic authentication through a combination of secret key and strong encryption. This ensures message integrity and data confidentiality. Think of a secret key as a password shared between the client and the authentication server. The encryption is performed through symmetric keys, using DES (Data Encryption Standard) or triple DES. Now there is support for AES (Advanced Encryption Standard) [2], although not implemented everywhere yet.

Kerberos is meant to provide authentication for interactive services like telnet, ftp or pop … where the user is prompted for a password and must login in real time. Symmetric key encryption allows real time authentication because it is a fast mechanism, the same key is used to encrypt and to decrypt the message.

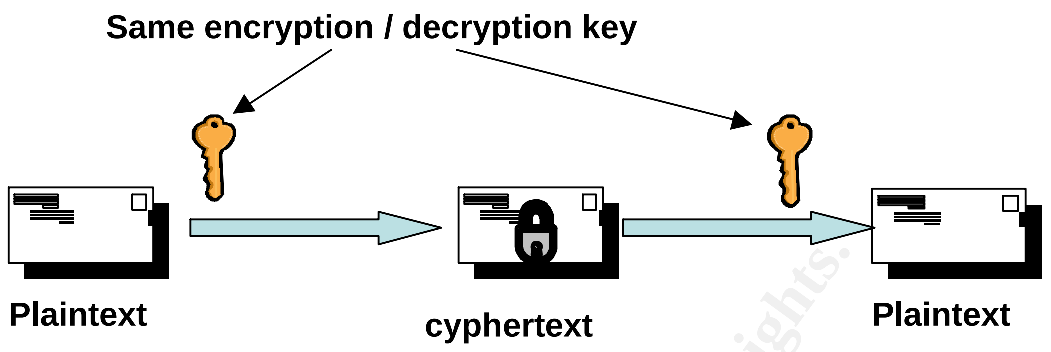


Fig 1. Symmetric key encryption

The Key Distribution Center

Kerberos protocol is used to authenticate "principals". A principal can be a simple user, an application server or any other network entity that needs to be authenticated.
Three parties are involved in the authentication process:

1) the client -or principal-
2) the server -or verifier-
3) the Kerberos server, called KDC (Key Distribution Center).

Let's consider a client that wants to connect to an application server using Kerberos. The KDC is trusted by both parties, and shares a secret key with each of them. Prior to any negotiation, secret keys or passwords from each principal have to be entered in the KDC. Encrypted in a local database of the KDC, this key is used to prove the principal's identity, and to establish an encrypted session between the KDC and the principal.
In exchange, the KDC will deliver a Ticket, required by the application server (verifier) to validate the principal's identity.

As we just saw, the KDC has two roles: the *Authentication Service* (AS) and the *Ticket Granting Service* (TGS). The Authentication Service exchange is done only once between a principal and the KDC. The KDC then delivers a Ticket Granting Ticket (TGT) through the TGS, that the client will use to obtain additional tickets. If the client wants to connect to multiple application servers, it will authenticate only once to the KDC. Then it will use the TGT he obtained to request further tickets to each application server, through the ticket granting service.

The Authentication Service

The first role of the KDC is the Authentication Service. The client (principal) initially requests a ticket to the KDC by giving its name, an expiration time until when the authentication will remain valid, the service required (tgs) and some other information, not mentioned here for clarity.

The KDC, if it finds the principal in its database, replies with two items:

- a client ticket containing a session key $S_{A,KDC}$, the expiration time and its tgs service name, all encrypted with the secret key of the principal K_A. The expiration time, typically a business day or eight hours, gives a time period during which the ticket will be valid.

- a granting ticket containing a session key $S_{A,KDC}$, the expiration time and the client's name, all encrypted with the secret key of the KDC K_{KDC}. This is what is known as the Ticket Granting Ticket. The principal, unable to decrypt the TGT, will use it later to request tickets to other services. As it is encrypted, the client can't read the data inside. If he tries to modify it, the KDC will not be able to decrypt it and it will be rejected.

Note that no password has been sent over the network in clear text. The password K_A is used locally by the KDC to encrypt the ticket and locally by the principal to decrypt it (remember the symmetric key: same key is used to encrypt and to decrypt a message). The principal and the KDC now share a session key $S_{A,KDC}$, created dynamically by the KDC, that they can use to encrypt their communications. The principal will prove its identity to the KDC by using this key, since he is the only one who can decrypt the preceding ticket.

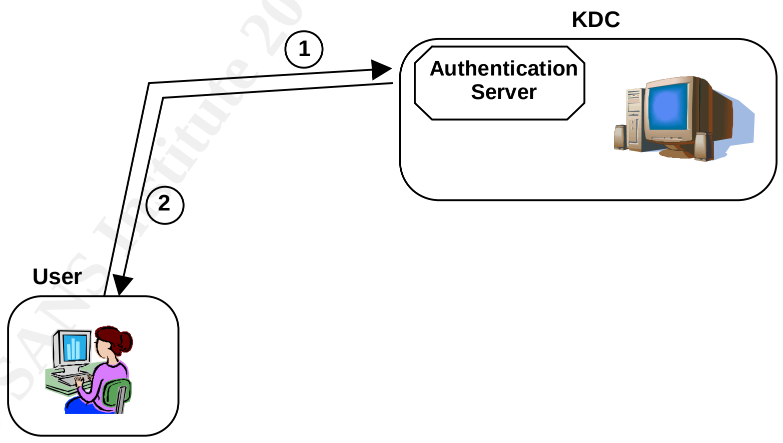


Fig 2. Authentication service. Ref [4]

1. **AS_REQ – {client name, expiration time, tgs service name, …}**
2. **AS_REP – { $S_{A,KDC}$, expiration time , tgs service name, …}. K_A**
   + { $S_{A,KDC}$, expiration time , client name, …}. K_{KDC}.

The Ticket Granting Service

The second role of the KDC is to distribute tickets, it is called the Ticket Granting Service. Once authenticated, the client that requests a particular application like telnet or ftp first asks the KDC. It doesn't query the application server directly. This request to the KDC contains several fields:

- an authenticator composed of : a timestamp and a checksum encrypted with the session key $S_{A,KDC}$ obtained earlier, shared between the client and the KDC. This proves the client's identity since he is the only one to know this session key. The checksum proves the message wasn't modified while transiting. The timestamp assures the message is recent, and is used to prevent "replay" attacks, since anyone could intercept the data over the network and use it later. Typically, the KDC must reply within five minutes for the message to be accepted. This is why it is important to have a good time synchronisation across your network when implementing Kerberos authentication. Consider using a protocol such as NTP (Network Time Protocol) to keep it accurate.

- the Ticket Granting Ticket received during the authentication exchange with the KDC. It is used by the KDC to check the client's name. If the client name present in the TGT doesn't match with the associated session key, this means the client has been impersonated, and the KDC is unable to decrypt the authenticator. Also, the KDC verifies the validity of the ticket by checking the expiration time of the authentication.

- the application service name to which the client wants to establish a connection.

- an expiration time for the Ticket Granting Ticket.

The KDC replies to the client (principal) with two tickets:

- the client ticket containing a new session key $S_{A,B}$ that the client and the application server will use to verify each other's identity and to encrypt their sessions. The ticket also encloses the application service name, and the expiration time of the new ticket. All these items being encrypted with the key $S_{A,KDC}$ shared between the KDC and the client, only known to the client.

- the server ticket containing the same session key $S_{A, B}$ as above, the client's name and the expiration time of the ticket. The server ticket being encrypted with the application server's secret key K_B, only known to the server.

It is then the responsibility of the client to forward this server ticket to the application server.

So, in order for the client to request access to the application server, it must first decrypt the client ticket and extract the session key $S_{A,B}$. Once extracted, the client uses this key to encrypt his authenticator, composed of a timestamp and a checksum. So the client sends this encrypted authenticator and the server ticket to the application server. Note that the application server does not have the session key $S_{A,B}$ yet. It will get it only if it is able to decrypt the ticket accompanying the authenticator, which is the server ticket. It was sent by the KDC to the client, encrypted with the application server secret key K_B, and is now forwarded by the client to the application server. As it is encrypted, no one except the application server is able to see what this ticket contains, not even the client. This is how the application server receives the session key $S_{A,B}$ to verify the client's identity and to share with it. It also verifies the validity of the ticket by checking the expiration time enclosed in the server ticket.

Optionally, the application sever replies to the client with a timestamp encrypted with their session key $S_{A,B}$. This is how the client verifies and validates the identity of the server; since the client and the server are the only one to know this session key. Again, the timestamp is used to prove the message is recent, and that it is not an old packet being resent.

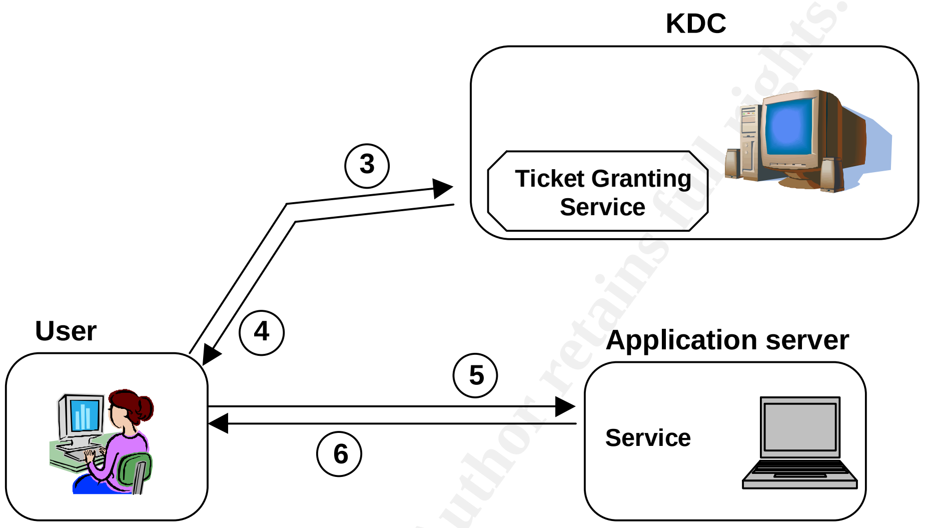


Fig 3. Ticket Granting Service. Ref [4]

3. TGS_REQ – {timestamp, checksum, …}.$S_{A,KDC}$
+ { $S_{A,KDC}$, expiration time , client name, …}. K_{KDC}.
+ application service name
+ expiration time
4. TGS_REP – {$S_{A,B}$, application service name, expiration time, …}. $S_{A,KDC}$
+ {$S_{A,B}$, client name, expiration time, …}. K_B
5. AP_REQ – {timestamp, checksum, …}.$S_{A,B}$
+ {$S_{A,B}$, client name, expiration time, …}. K_B
6. AP_REP – {timestamp}.$S_{A,B}$

Summary

So to resume, we have:

A client possessing
- his secret key K_A to share with the KDC, used to extract the session key $S_{A,KDC}$.
- the session key $S_{A,KDC}$.
- the TGT

$\left.\right\}$ To request further tickets to the KDC

- the session key $S_{A,B}$ to prove its identity to the application server and to exchange encrypted messages.

A server possessing

- his secret key K_B to share with the KDC, used to extract the session key $S_{A,B}$.
- the session key $S_{A,B}$ to prove its identity to the client and to exchange encrypted messages.

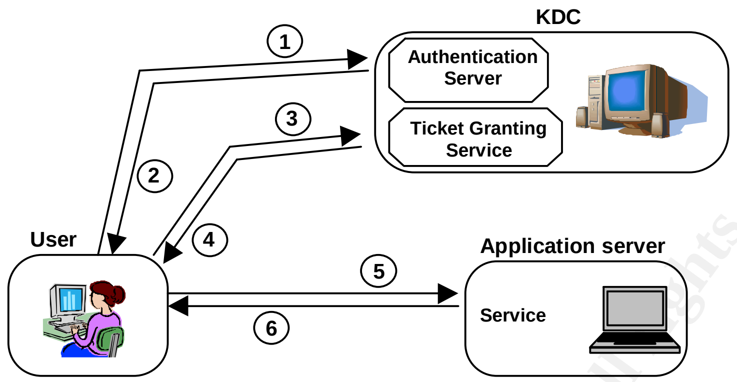


Fig 4. Kerberos authentication. Ref [4]

There is now mutual trust between the client and the application server: they are **authenticated**.

Realms

Much like the Microsoft Windows domains, Kerberos has the ability to divide the network in groups or “realms”. There is at least one KDC by realm, and optional slave KDCs. This separation is made to avoid too many requests being sent to a single KDC, which would become a bottleneck for the authentication service and thus for the whole network. A realm name is typically mapped to the Domain Name of the network (DNS), or to sub-domain names, although this is not mandatory. Kerberos refers to principals (item, user or service, which needs to be authenticated) and realms under a common notation:

principal/instance@REALM.COM

where the instance is an optional entry for the principal in the KDC’s database. Note that the realm is commonly written in capital letters to differentiate the Domain Name from the Kerberos realm. A principal located in one realm can contact a server located in another realm using the cross-realm authentication capability of Kerberos v5:

1. The client first asks his local KDC for a Ticket Granting Ticket for the remote Kerberos KDC, just as it would do it the request any other service. The remote KDC being the KDC of the realm, which the application server belongs to.
2. Then, the client forwards the TGT it obtained to the remote KDC, requesting a ticket to access an application server in this remote realm.
3. Finally, the client sends its authenticator and ticket to the application server in the remote realm, and connects to it.

Naturally, network connectivity is required between the client and all KDCs, and between the client and the application server. This is a very short description summary of cross realm authentication, as in reality, it is much more complex. Generally, a KDC will share an inter-realm key with its hierarchical KDC, based on the name of the realm. For example LOC1.VOILA.COM will share an inter-realm key with VOILA.COM … and so on. Direct links can also be created to be more effective, which means LOC1.VOILA.COM can share an inter-realm key directly with LOC2.VOILA.COM without having to make a request to VOILA.COM. It can also share a direct inter-realm key with SANS.ORG, which will greatly improve the authentication process.

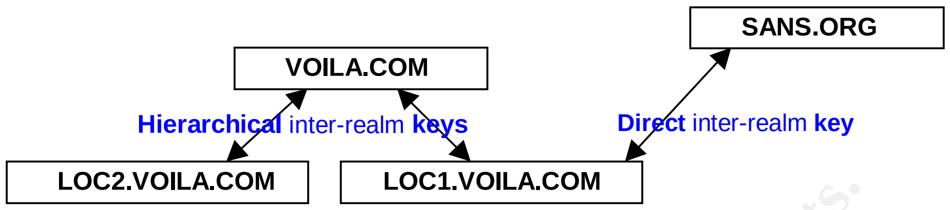


Fig 5. Inter-realm keys

In order for a principal to reach the local KDC (the KDC that belongs to his realm), there are two different methods. The first one is through a configuration file on the client device, where the realm name is mapped to the IP address or hostname of the KDC. This is the method used by default.

```
[realms]
    LOC1.VOILA.COM = {
        kdc = toto.loc1.voila.com:88
        admin_server = toto.loc1.voila.com:749
        default_domain = loc1.voila.com
     }
```

The second method, available with Kerberos v5, is to configure the Domain Name Server with the required information. A special TXT record in the DNS configuration file, beginning with "_kerberos", will indicate the name of the realm. Additionally, a special SRV record also beginning with "_kerberos" will mention the hostname of the KDC and TCP/UDP port numbers to use. It is recommended to add an alias with the name of "kerberos" for the KDC, using the CNAME record.

```
$ORIGIN loc1.voila.com.
_kerberos                TXT        " LOC1.VOILA.COM"
kerberos                 CNAME      toto
_kerberos._udp           SRV        0 0 88 toto
_kerberos-master._udp    SRV        0 0 88 toto
_kerberos-adm._tcp       SRV        0 0 749 toto
_kpasswd._udp            SRV        0 0 464 toto
```

Kerberos connectivity

As we saw through the configuration files, Kerberos uses several TCP and UDP ports for its communications. Network connectivity needs to be set up correctly for the KDC to be able to deliver tickets, and for administration purposes. In particular, firewalls need to be configured according to your environment, to allow Kerberos traffic. It is recommended to define one rule per TCP/UDP port allowed through for clear and granular management, but also for logging purposes.

The ports that Kerberos v5 needs are:

- UDP port 88, between the KDC and the principal. Used for initial ticket request. Being connectionless, UDP is well suited for small messages. However, as we will see later, Microsoft uses a special field in the Kerberos protocol for authorization. This field containing much data enclosed, TCP transport protocol is used instead of UDP by Microsoft. MIT also allows the use of TCP connections for compatibility with Microsoft.

- TCP port 749, to the KDC. Used to change passwords on UNIX systems, and for administration purposes.

- UDP/TCP port 464, to the KDC. Used to change passwords on Microsoft systems.

Then, each service using Kerberos authentication requires its own ports. Here are a few examples of communication ports between a client and a server using Kerberos:

21 / TCP	ftp	Kerberos ftp (like regular ftp)
23 / TCP	telnet	Kerberos telnet (like regular telnet)
543 / TCP	Klogin	Kerberos login
544 / TCP	Kshell	Kerberos remote shell
545 / TCP	Ekshell	Encrypted kshell
1109 / TCP	Kpop	Kerberos POP
2053 / TCP	Knetd	Kerberos de-multiplexer
2105 / TCP	Eklogin	Encrypted klogin
…		

Flags

To increase the functionalities of the authentication protocol itself, Kerberos includes several flags used to simplify its usage and make it more powerful and transparent.

- **pre-authentication** tickets: this flag is set when a principal gives more information than traditional password to the KDC to prove its identity. This could be biometric authentication for example, or the use of a smart card. It that case, we are sure the client authenticating is who he claims to be, even if his password has been compromised.

- **renewable** ticket: applications that need to authenticate again after the ticket expiration time (typically eight hours), can ask for a ticket to be renewed instead of requesting for a new one.

- **proxy** tickets: applications can request a ticket for another application or on behalf of a user. This is very useful, and particularly in Windows 2000 environment, where system services often need to act on behalf of a user. This could also be a web service requesting access for a database server.

- **forwarded** tickets: applications or users can request a new ticket to the KDC, based on a previously obtained ticket with the forwardable flag set. This option of the Kerberos protocol is what makes it possible to implement single-sign-on (SSO) using Kerberos. Based on the obtainment of a single ticket, a user could request access to all services across the network, without having to re-authenticate. It is one of the greatest improvement Kerberos can add to your network, because it means

- more transparency to the user, the user logs in only once, has only one password to remember.
- simplified administration, there is one central location for administrating users.

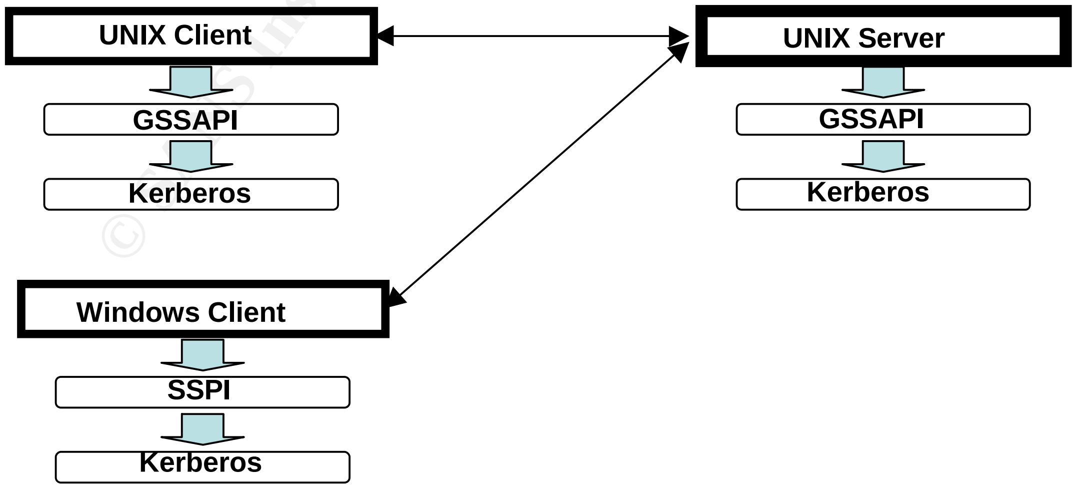
Kerberos integration

At its beginning, Kerberos was mostly used in Universities, Colleges and in financial organisations. Since its integration as default authentication mechanism in Microsoft Windows 2000, it has become more popular.

1) First, of course, because Microsoft had decided to use it. Many people were curious about it and took a closer look at Kerberos.
2) Then, as network environments were evolving, hacking tools becoming more and more popular and easy to use, security took a higher place in network administration. Many organisations were willing to add secure authentication to their environment, and Kerberos made this possible.
3) Finally, with its 20 years of existence, Kerberos protocol has come to a robust, flexible package, that can easily be integrated into any application server or client.

The GSSAPI

This integration is often performed through an API: the GSSAPI (Generic Security Services Application Program Interface). This is an abstraction layer above Kerberos 5 defined in RFC 1964 [14], used by applications to provide security services. Standard across all UNIX platforms, it is used as a link between an application and, in our case, Kerberos. Similar to the GSSAPI, the SSPI (Security Support Provider Interface) was developed by Microsoft fro Windows 2000 and above. Fortunately the GSSAPI and the SSPI are fully compatible.



GSSAPI / SSPI

"Kerberized" applications

Provided these additional layers, any application can be "kerberized", i.e. turned into a Kerberos aware application. Today, this is not a standard yet, but more of an additional module used to patch the rough application. However, you will see that many clients and servers take advantage of it. As a distributed authentication mechanism, both the client and the server have to be kerberized.

Here is a non exhaustive list of applications actually offering Kerberos authentication, whether natively, through compilation options, or with an additional module or plugin:

- *standard remote connection servers, **telnet**, **ftp**, **rlogin**, **rshell**… are available:*

○ Redhat 9 packages: URL: http://www.redhat.com (27 Nov. 2003).
krb5-workstation-1.2.7-10.i386.rpm
krb5-server-1.2.7-10.i386.rpm

○ Debian packages: URL: http://www.debian.org (27 Nov. 2003).
krb5-clients - Secure replacements for ftp, telnet and rsh using MIT Kerberos
krb5-ftpd - Secure FTP server supporting MIT Kerberos
krb5-rsh-server - Secure replacements for rshd and rlogind using MIT Kerberos
krb5-telnetd - Secure telnet server supporting MIT Kerberos

○ Sun: URL: http://wwws.sun.com/software/security/kerberos/ (27 Nov. 2003).
SEAM Kerberized telnet, rcp, rsh, ftp, rlogin

○ MIT offers clients and servers for all platforms Windows, Irix, Solaris, Macintosh, Linux: URL: http://web.mit.edu/kerberos/dist/index.html (27 Nov. 2003).

○ Connectivity Kerberos: URL: http://www.hummingbird.com/ (27 Nov. 2003).
○ Nifty Telnet: URL: http://andrew2.andrew.cmu.edu/dist/niftytelnet.html (27 Nov. 2003).
○ KFTP: URL: http://andrew2.andrew.cmu.edu/dist/kftp.html (27 Nov. 2003).
○ Better Telnet: URL: http://www.cstone.net/~rbraun/mac/telnet/kerberos.html (27 Nov. 2003).
○ TN3270: URL: ftp://ftp.brown.edu/pub/mac/tn3270/ (27 Nov. 2003).
○ Datacomet: URL: http://www.databeast.com/ (27 Nov. 2003).
○ Reflection Secure: URL: http://www.wrq.com (27 Nov. 2003).
○ Yet Another FTP Client (yafc): URL: http://sourceforge.net/projects/yafc/ (27 Nov. 2003).
○ FileZilla: URL: http://sourceforge.net/projects/filezilla/ (27 Nov. 2003).
○ …

- ***email** clients and servers:*
○ Eudora: URL: http://www.eudora.com/ (27 Nov. 2003).
○ Pine: URL: http://www.washington.edu/pine/ (27 Nov. 2003).
○ UW IMAP: URL: http://www.washington.edu/imap/ (27 Nov. 2003).
○ Microsoft Outlook: URL: www.microsoft.com/office/outlook/default.asp (27 Nov. 2003).

- ***SSH** secure shell:*
○ SSH: URL: http://www.ssh.com (27 Nov. 2003).
○ OpenSSH: URL: http://www.openssh.com/ (27 Nov. 2003).

- ***PAM** (Pluggable Authentication Module):*
  - pam_krb5: URL: http://sourceforge.net/projects/pam-krb5/ (27 Nov. 2003).
  - Debian package: libpam-heimdal - PAM module for Heimdal Kerberos 5
    libpam-krb5 - PAM module for MIT Kerberos
  - RedHat package: pam_krb5-1.60-1.i386.rpm
  - HP: URL:http://www.software.hp.com/portal/swdepot/displayProductInfo.do?productNumber=J5849AA (27 Nov. 2003).
  - Kerberos 5 PAM: URL: http://www.fcusack.com/ (27 Nov. 2003).
  - Kerberos 5 PAM: URL: http://is.rice.edu/~wymanm/projects/ (27 Nov. 2003).

- ***NFS** (Network File System):*
  - SEAM Kerberized NFS: URL:http://wwws.sun.com/software/security/kerberos/ (27 Nov. 2003).
  - NFS: URL: http://sourceforge.net/projects/nfs/ (27 Nov. 2003).

- ***AFS** (Andrew File System):*
  - CSL AFS/Kerberos: URL: http://www.cs.wisc.edu/csl/doc/info/nt-software/afsclient (27 Nov. 2003).
  - Arla: URL: http://www.stacken.kth.se/projekt/arla/ (27 Nov. 2003).
  - OpenAFS : URL: http://www.openafs.org/doc/index.htm (27 Nov. 2003).

- ***DCE** (Distributed Computing Environment):*
  - Open Group: URL: http://www.opengroup.org/dce/ (27 Nov. 2003).
  - Entegrity: URL: http://www2.entegrity.com/products/dce/prod_eval.shtml (27 Nov. 2003).

- ***Web authentication**:*
  - Mozilla has a Kerberos plugin: URL: http://www.mozilla.org (27 Nov. 2003).
  - Apache also has a plugin: URL: http://www.apache.org/ (27 Nov. 2003).
  - Microsoft IIS and IE: URL: www.microsoft.com/iis (27 Nov. 2003).

- ***Database** applications:*
  - Oracle: URL: http://www.oracle.com/ (27 Nov. 2003).
  - DB2: URL: http://www-3.ibm.com/software/data/db2/ (27 Nov. 2003).
  - MySQL: URL: http://www.mysql.com/ (27 Nov. 2003).

- ***WiFi** wireless technologies:*
  - Symbol uses Kerberos as its standard security mechanism: URL: http://www.symbol.com/ (27 Nov. 2003).

- ***Java**:*
  - Java Secure Socket Extension (JSSE) now includes Kerberos support. URL: http://java.sun.com/j2se/ (27 Nov. 2003).

- *and many more…*

Kerberos implementations

The Kerberos implementation we talked the most about in this document is the one from the Massachusetts Institute of Technology. Other organizations and commercial vendors have created their own implementations.

- Cygnus solutions developed KerbNet. The latest version was v1.2, but it doesn't seem to be maintained anymore. It used to be URL: http://www.cygnus.com/ (27 Nov. 2003).

- OpenVision created the reference implementation of the GSSAPI for Kerberos Version 5. URL: http://www.ov.com/ (27 Nov. 2003).

- Cisco has integrated the client portion of Kerberos in its IOS.

- Cybersafe developed the first commercial version of Kerberos. URL: http://www.cybersafe.ltd.uk/products.htm (27 Nov. 2003).

- The Center for Parallel Computers in Sweden has a free Kerberos solution called Heimdal. URL: http://www.pdc.kth.se/heimdal (27 Nov. 2003). Current release is v0.6.

- Sun corporation developed SEAM (Sun Enterprise Authentication Mechanism). URL: http://wwws.sun.com/software/security/kerberos/ (27 Nov. 2003). Current release is version 1.0.1 for Solaris 8.

- GNU has a -still incomplete- free implementation of Kerberos called Shishi. URL: http://www.gnu.org/software/shishi/ (27 Nov. 2003). Current release is version 0.0.8.

- Microsoft implementation of Kerberos.
  URL: http://www.microsoft.com/technet/security/news/kerb2000.asp (27 Nov. 2003).

- MIT implementation. URL: http://web.mit.edu/kerberos/www/ (27 Nov. 2003). Current release is v1.3.1 for UNIX, v2.5 for Windows, and v5 for Macintosh.

Kerberos, a good base for single sign-on

As you can see, Kerberos is present and supported in many applications today. With its wide deployment and by fully enabling ticket forwarding, Kerberos has become a real enabler for single sign-on (SSO) solutions. While it still requires extra measures to put in place, Kerberos can be set up as a strong base to allow users to login only once in the network and access transparently all applications available.
Microsoft has started to use Kerberos, and its forwarding tickets capability, allowing authenticated users to access printing services, CIFS/SMB, LDAP, IPSec, QoS management, etc…transparently, although Microsoft has slightly enhanced the specifications of Kerberos. They have used a previously unused field for authorization, which is critical for single sign-on user management. Developed aside from Kerberos RFC 1510, they created a proprietary authorization field, which enables a Windows KDC (often a Domain Controller) combined with LDAP (Lightweight Directory Access Protocol) to manage all access controls from a central location. They made single sign-on possible with authentication and authorization, granular enough to separate user

rights, depending on the service they require. Now, Microsoft has publicly released this code allowing authorization, but there is no standard, as they use information specific to Windows architecture. MIT and other Kerberos implementations don't support authorization yet.

But there is another way to add authorization to Kerberos. The current release of Java Standard Edition (J2SE) implements a service called JAAS Java Authentication and Authorization Service which, combined with Java GSSAPI, can provide a good way to control user login. Although it is not directly included in the Kerberos protocol, this authorization mechanism works and can be quit powerful. JAAS is based on PAM (Pluggable Authentication Module) and is independent from the underlying security layer, Kerberos in our case. Here is a good paper explaining the use of Kerberos in Java [3].

Limitations of Kerberos

A last point on the limitations of Kerberos protocol:

Although it provides strong encryption over the network, it cannot prevent password guessing attacks. A weak password is always a risk, even though it is not transmitted in clear over the network. It is advised to use a "passphrase", as opposed to a password.

As we saw, Kerberos provides secure authentication. Authorization and accounting are not part of the protocol specifications. Even if Microsoft uses its own authorization field, still an additional layer has to be added to provide a strong, complete, centralized authentication protocol.

If both Kerberos and non-Kerberos applications reside on the network, administrators should be careful not to allow users to authenticate using their Kerberos password for other applications. The deployment of a strong encryption authentication mechanism would become useless.

Each network service needs to be adapted to Kerberos. Often, an additional module, plugin, or code modification is required to have a kerberized application. It has not become a standard yet.

Kerberos needs a fully time synchronized network, tickets being time based. If it is not the case, authentication becomes impossible and network services unavailable.

Kerberos bases its security on a single point of failure, the KDC. If the KDC is compromised, then your entire network is. If the KDC is unavailable, your network is also unavailable. The scalability is an important point to consider. Also, the KDC must be a dedicated, secured, protected server configured with minimal access.

Like any other application, Kerberos has its vulnerabilities. Buffer overrun, Denial Of Service, etc… are also affecting Kerberos. Kerberos administrator has to pay attention to new vulnerabilities and related patches to be applied.

Conclusion

Providing additional security to a network environment is always a good thing. Authentication has long been left apart from security. Administrators used to focus on first protecting their perimeter by implementing firewalls, forgetting the internal network or considering it was secured enough, since it was separated from outside. It took a long time for Kerberos to be recognized and used by IT professionals. Its use as default authentication protocol in Windows 2000 has contributed to this recognition.
Although it requires a complete integration in the network architecture to be efficient, Kerberos provides a secure and simple method to administer authentication mechanism. With its deployment in most popular applications, it can now be set up as a base for single sign-on for all infrastructures.

References

1. Kohl, J. Neuman, C. "The Kerberos Network Authentication Service (V5)". September 1993. URL: http://www.ietf.org/rfc/rfc1510.txt (27 Nov. 2003).
2. Chown, P. « Advanced Encryption Standard (AES) ». June 2002. URL: http://www.ietf.org/rfc/rfc3268.txt (27 Nov. 2003).
3. Upadhyay, M. Marti, R. "Single Sign-on Using Kerberos in Java". May 2001. URL: http://java.sun.com/j2se/1.4.2/docs/guide/security/jgss/single-signon.html (27 Nov. 2003).
4. "Kerberos: The Network Authentication Protocol". URL: http://web.mit.edu/kerberos/www/ (27 Nov. 2003).
5. Kohl, J. Neuman, C, Ts'o, T. "The Evolution of the Kerberos Authentication Service". 1994. URL: ftp://athena-dist.mit.edu/pub/kerberos/doc/krb_evol.lpt (27 Nov. 2003).
6. "Windows 2000 Kerberos Authentication". July 1999. URL: http://www.microsoft.com/windows2000/techinfo/howitworks/security/kerberos.asp (27 Nov. 2003).
7. Viard, R. "Kerberos". May 2001. URL: http://www.easter-eggs.org/article_105_Kerberos.html (27 Nov. 2003).
8. "Kerberos FAQ". V2.0. Aug. 2000. URL: http://www.faqs.org/faqs/kerberos-faq/general/ (27 Nov. 2003).
9. Backman, D. "Kerberos Network Design Manual". URL: http://www.networkcomputing.com/netdesign/kerb1.html (27 Nov. 2003).
10. "Kerberos V5 Installation Guide". v1.2. Jan. 2002. URL : http://www.lns.cornell.edu/public/COMP/krb5/install/install_toc.html (27 Nov. 2003).
11. Conry-Murray, A. "Kerberos: Computer Security's Hellhound". May 2001. URL: http://www.networkmagazine.com/article/NMG20010620S0008 (27 Nov. 2003).
12. Garman, J. "Single Sign-on for Your Web Applications with Apache and Kerberos". Nov. 2003. URL: http://www.onlamp.com/pub/a/onlamp/2003/09/11/kerberos.html (27 Nov. 2003).
13. Smith, D. "Implementing Kerberos". Dec. 2001. URL : http://www.samag.com/documents/s=1769/sam0112d/0112d.htm (27 Nov. 2003).
14. Linn, J. "The Kerberos Version 5 GSS-API Mechanism ». June 1996. URL: http://www.ietf.org/rfc/rfc1964.txt (27 Nov. 2003).