

Cross-Platform Authentication & Single Sign-On

*Covers
Unix and Windows*



Kerberos

The Definitive Guide

O'REILLY®

Jason Garman

Kerberos

The Definitive Guide

Other computer security resources from O'Reilly

Related titles	802.11 Security Building Internet Firewalls Computer Security Basics Java Cryptography Java Security Linux Security Cookbook Secure Programming Cookbook for C and C++ Network Security with OpenSSL Practical Unix and Internet Security	Secure Coding: Principles & Practices Securing Windows NT/2000 Servers for the Internet SSH, The Secure Shell: The Definitive Guide Web Security, Privacy, and Commerce Database Nation Building Secure Servers with Linux
-----------------------	---	---

Security Books Resource Center

security.oreilly.com is a complete catalog of O'Reilly's books on security and related technologies, including sample chapters and code examples.



oreillyn.com is the essential portal for developers interested in open and emerging technologies, including new platforms, programming languages, and operating systems.

Conferences

O'Reilly & Associates bring diverse innovators together to nurture the ideas that spark revolutionary industries. We specialize in documenting the latest tools and systems, translating the innovator's knowledge into useful skills for those in the trenches. Visit conferences.oreilly.com for our upcoming events.



Safari Bookshelf (safari.oreilly.com) is the premier online reference library for programmers and IT professionals. Conduct searches across more than 1,000 books. Subscribers can zero in on answers to time-critical questions in a matter of seconds. Read the books on your Bookshelf from cover to cover or simply flip to the page you need. Try it today with a free trial.

Kerberos

The Definitive Guide

Jason Garman

Troubleshooting

Working with Kerberos can seem like an exercise in futility. The combination of complex software, interoperation between Kerberos implementations and diverse operating systems, and terse (at best) error messages tend to bring premature balding to the administrators responsible for the smooth operation of their systems. However, working with Kerberos does not have to be this frustrating.

To provide a systematic approach to solving problems in Kerberos, this chapter begins with a discussion of the various debugging tools and techniques, including a sample decision tree to follow when facing a new problem. Becoming familiar with these tools and techniques before disaster strikes will be helpful when the inevitable problem occurs. The second section will use those tools and techniques to diagnose some typical problems that occur in a Kerberos system. It will describe the symptoms of these common problems and then show possible solutions to solve them.

A Quick Decision Tree

So you're having a problem with your Kerberos installation. The first step to solving this problem, like debugging any other issue, is to narrow down the root cause. We'll determine if the problem falls into three distinct categories, and continue our analysis from there.

An easy way to categorize an error involving Kerberos authentication is through what tickets the client can acquire for a service. Let's look at the three top-level categories:

Client can't get an initial Ticket Granting Ticket. This is most likely a client-specific problem, especially if logging in with the user principal and password works on other clients. Of course, it could also mean that the password entered for the user principal is incorrect.

The most likely culprits include time-synchronization problems and issues reaching the Kerberos server due to misconfiguration of the client. It is also possible that the client does not share a compatible encryption type for the users'

secret key with the KDC. This can happen, for example, when attempting to interoperate between a Unix client and a Windows domain controller. By default, Windows domain controllers create user entries with an RC4-HMAC encryption type, which most Unix Kerberos implementations do not understand. Newer versions of Heimdal and MIT Kerberos 5 will support this encryption type.

Client has valid TGT but gets error before a service ticket is acquired. Once again, this is most likely a problem with the client. The usual suspects in this scenario include Kerberos misconfiguration on the client, which we'll cover in a subsequent section.

Another possibility is that the service principal that the client requested simply does not exist. Examining the KDC log files is a good way to determine if the client is reaching the KDC, and if so, if it is attempting to acquire tickets for a principal that does not exist, or is in a different realm.

Client has valid TGT and service ticket, but reports error on connection to Kerberized service. There is most likely a problem with the server. At this point, the client has received a ticket for the service and presented it to the service for authentication. The most common cause for a failure at this point is a mismatch in the encryption types or key version numbers for the Kerberos service between the service's keytab and the KDC. The KDC may have issued a ticket with an encryption type that the service did not understand, or perhaps the service's keytab contains an incorrect encryption key. The service may not be able to read its keytab at all; ensure that the service's keytab is readable by the user the service runs as, and that the appropriate configuration is in place to point the service to the location of the keytab.

Another possibility is that the server's DNS or Kerberos configuration file is not configured correctly. Make sure that the target server's hostname can be correctly resolved by using diagnostic tools such as ping and nslookup.

General root causes that should be investigated include time synchronization and correct hostname and DNS settings on all machines.

Finally—this may seem obvious—it is easy to overlook the final step: recording the error and the solution that was used to solve the problem. All too often, especially with some of the more esoteric errors that Kerberos can produce, administrators experience an error one day, solve the problem, then experience the same error a week later, and have to track down the root cause again, only to find that the solution is vaguely familiar.

Debugging Tools

The MIT Kerberos distribution includes a small sample Kerberized client/server application. These example applications are located in the `src/appl/sample` subdirectory of the MIT Kerberos 5 distribution.

Just like any other Kerberized server, the sample server requires a service principal and access to the secret key associated with that principal through a keytab file. By default, the sample server uses a principal name of “sample,” with an instance of the hostname that it is running on. If you’re having trouble with a particular service principal, the sample server and client can use any principal name to communicate with each other, given the sample server has read access to the service’s keytab file.

The command-line arguments accepted by the sample server are:

```
> ./sserver -h
usage: ./sserver [-p port] [-s service] [-S keytab]
```

The `-p` argument specifies what TCP port that the server will listen on for client requests. If this argument isn’t specified, then `sserver` will immediately exit. The `-s` option can be used to specify a particular service principal (instead of the default, “sample”). For example, the host principal can be specified by `-s host`. Finally, the `-S` option specifies a keytab file in which the server can find the secret key for the service principal. By default, `sserver` will use `/etc/krb5.keytab`.

Ensure that a valid keytab entry for the principal you’re using to test exists in a keytab file and is readable by the user you’re starting `sserver` as. Note that the server won’t test for the readability of the keytab until a client connects to it, and your client will report “Permission denied”.

The command-line arguments that the client accepts are similar:

```
> ./sclient
usage: ./sclient <hostname> [port] [service]
```

A successful exchange looks like the following, assuming that you have shells open on both of the hosts `freebsd` and `slave`, and their prompts are `freebsd>` and `slave>` respectively:

```
freebsd> ./sserver -p 8888 -s sample -S /tmp/sample.keytab
slave> ./sclient freebsd 8888 sample
sendauth succeeded, reply is:
reply len 27, contents:
You are jgarman@WEDGIE.ORG
```

If you choose a service name other than “sample,” specify the service name on the command lines to both the server and the client. Just like with any other Kerberos client/server application, you’ll see that you now have Kerberos tickets for the sample service principal:

```
client> klist
Ticket cache: FILE:/tmp/krb5cc_p27758
Default principal: jgarman@WEDGIE.ORG

Valid starting    Expires          Service principal
02/26/03 02:34:47  02/26/03 10:58:19  krbtgt/WEDGIE.ORG@WEDGIE.ORG
02/26/03 02:35:51  02/26/03 10:58:19  sample/freebsd.wedgie.org@WEDGIE.ORG
```

A similar set of programs exists for Heimdal Kerberos. Heimdal actually includes several sample client/server applications, and the two client/server sets that are the most useful for our debugging purposes are the `tcp_client/tcp_server` applications and the `gssapi_client/gssapi_server` applications, both found in the `appl/test` directory inside of the Heimdal source distribution. The `tcp_client` and `tcp_server` applications use the Kerberos 5 API directly, and the `gssapi_client` and `gssapi_server` applications are simple applications that utilize the GSSAPI for authentication.

The command-line arguments for these programs are similar to the MIT Kerberos sample applications:

```
> ./gssapi_server -h
Usage: gssapi_server [-fh] [--port=port] [-p port] [--service=service]
      [-s service] [--keytab=keytab] [-k keytab] [--fork] [--help] [--version]
-p port, --port=port      port to listen to
-s service, --service=service service to use
-k keytab, --keytab=keytab keytab to use
-f, --fork                do fork
> ./gssapi_client -h
Usage: gssapi_client [-fh] [--port=port] [-p port] [--service=service]
      [-s service] [--keytab=keytab] [-k keytab] [--fork] [--help] [--version] host
-p port, --port=port      port to listen to
-s service, --service=service service to use
-k keytab, --keytab=keytab keytab to use
-f, --fork                do fork
```

The arguments accepted by `tcp_client` and `tcp_server` are the same as the `gssapi_client` and `gssapi_server` applications, respectively. Unlike the MIT Kerberos sample applications, the default principal name used by the Heimdal testing applications is “test”.



When testing these programs, it was noted that the `keytab` command-line option did not function correctly; instead, the application continued to try and access `/etc/krb5.keytab` regardless of the filename passed through the `-k` parameter.

An example of successful output:

```
freebds> ./gssapi_server -s sample -p 8888
slave> ./gssapi_client -s sample -p 8888 freebsd
User is 'jgarman@WEDGIE.ORG'
gss_verify_mic: hej
gss_unwrap: hemligt
```

Another tool that can be helpful is the Kerberized telnet daemon. It has rather verbose output so that errors can be readily gleaned from the messages it prints when connecting. When creating the realm in Chapter 4, we set up a telnet daemon to test the new realm. Make sure that, when using telnet to test Kerberos functionality, you use the `-a` option on the client to tell it to automatically attempt Kerberos authentication. Also, ensure that the telnet client program is actually a Kerberized version, and not the system telnet that may not be Kerberized.

Errors and Solutions

With the debugging tools presented above, we'll run through a few problem scenarios, from the initial symptoms of a problem through to its solution.

Errors Obtaining an Initial Ticket

Several errors can occur when attempting to obtain an initial Ticket Granting Ticket from a Kerberos KDC. Since there are many ways to obtain a TGT, such as through integrated login with a PAM Kerberos module, the best way to narrow down problems is by using the Unix `kinit` program manually. This will work even if your KDC is a Windows domain controller, given that the principal you're testing has been set up for DES encryption (see Chapter 8).

Let's go through a few examples:

```
> kinit
Password for jgarman@WEDGIE.ORG:
kinit(v5): Preauthentication failed while getting initial credentials
```

If your realm requires pre-authentication (see Chapter 6), then this message is typically just Kerberos-speak for “incorrect password.” Note that Windows domain controllers require pre-authentication by default. Also note that this message can result from a client that does not support the pre-authentication type required by the KDC. However, all of the Kerberos implementations we cover here support the Encrypted Timestamp (PA-ENC-TIMESTAMP) pre-authentication method. Of course, if you are interoperating with a Kerberos implementation that does not support pre-authentication, and your realm requires it, you will have to disable pre-authentication in the KDC policy.

Next, there is a possibility that the KDC could not find an appropriate encryption key with which to encrypt the response. When a Kerberos 5 client contacts a KDC through the AS exchange for an initial Ticket Granting Ticket, the client sends a list of encryption types that it understands. If the KDC cannot find a secret key associated with one of the encryption types included in the request, it will return an error.

Encryption type mismatches can also occur later on in the Kerberos exchange, and we'll cover that in a later section. Errors obtaining an initial ticket can also be caused by hostname/DNS misconfiguration, or a missing or incorrect Kerberos configuration file. These possibilities will also be covered soon in a later section.

Finally, another common error that can cause the pre-authentication failed message is a clock synchronization problem (which can cause all sorts of other strange problems, as well), covered next.

Unsynchronized Clocks

Another common root cause of Kerberos problems is the lack of clock synchronization between all participating hosts. Usually the error message produced when there is a clock mismatch is self-explanatory. For example:

```
krb5_rd_req failed: Clock skew too great
```

It is recommended that all participating hosts in a Kerberos realm be synchronized to a central time source. The Network Time Protocol (NTP) fits this bill perfectly. NTP is discussed briefly in the “Before You Begin” section in Chapter 4, and more information can be found at the NTP home page at <http://www.ntp.org>.

Since NTP and Kerberos both use Universal Coordinated Time (UTC) to compare clocks, time zone differences do not affect the operation of Kerberos.

Incorrect or Missing Kerberos Configuration

Every client needs to know two things: the realm that every host it wishes to communicate with belongs to, and the KDCs that are responsible for those realms. Therefore, a client requires a mapping between domain names and realms, as well as realms and their KDCs. Traditionally, these mappings are hardcoded inside a configuration file, */etc/krb5.conf*, on Unix hosts. However, with the advent of Windows 2000, DNS realm and KDC mapping is becoming more popular, as Windows 2000 and above use DNS to find realm and KDC information by default.

Either way that your realm (and other realms your hosts may communicate with) handles its Kerberos configuration, incorrect or missing Kerberos configuration information will cause requests for tickets to fail, sometimes silently—giving no error messages to point toward the cause of failure, or worse yet, misleading error messages that give no indication of the root cause of failure.

Let’s begin with some simple examples from a Unix client running MIT. Our client has no */etc/krb5.conf* file. If there is a pre-existing credential cache (pointed to by the environment variable *KRB5CCACHE*, by default */tmp/krb5cc_UID* where *UID* is the Unix *UID* of the current user), then *kinit* will request a ticket as that principal from the KDC. When there is no pre-existing credential cache, MIT *kinit* forms the user principal by using the Unix username as the username portion of the principal, and the realm is determined by the *default_realm* variable in the *libdefaults* stanza of */etc/krb5.conf*. Since, in this case, there is no */etc/krb5.conf* file, *kinit* will complain:

```
> whoami
jgarman
> kinit
kinit(v5): Configuration file does not specify default realm when parsing name
jgarman
```

Now let’s create a minimalist */etc/krb5.conf* file:

```
[libdefaults]
  default_realm = WEDGIE.ORG
```

With this in place, MIT Kerberos can now determine the default realm. Note that even if a correct DNS domain name-to-realm mapping is available in DNS, MIT still requires a *default_realm* entry in */etc/krb5.conf* to work correctly. Also, without a default realm defined, some GSSAPI applications may also behave incorrectly, giving “generic error” messages.

Note that this minimalist */etc/krb5.conf* file will only work if you have correct DNS entries for your Kerberos realm, as discussed in “DNS and Kerberos” in Chapter 4. There has to be a way for the client to find its KDC, either through static configuration files or through DNS. If the client cannot find the KDC for a given realm, it will report back accurately:

```
kinit(v5): Cannot resolve network address for KDC in requested realm while getting
initial credentials
```



MIT Kerberos has a bug in which a KDC returns an error message claiming that it has no support for the requested encryption type to all ticket requests if the KDC has no */etc/krb5.conf* file. Creating a */etc/krb5.conf* file on the KDC, even if it is blank, will fix this problem.

Heimdal kinit also uses an existing credential cache file, if there is one, to determine the principal that kinit will acquire tickets for. If there is no */etc/krb5.conf* file, or if it does not specify a default realm, it will use the domain name component of the local hosts’ hostname as the realm. Heimdal will also consult DNS for DNS hostname-to-Kerberos realm mapping as well as KDC location, or use static configuration in */etc/krb5.conf*.

In all cases, static configuration in */etc/krb5.conf* will short-circuit the location of KDCs and domain-realm mappings. If static information is located in the configuration file, the static entries will override any DNS information available.

If issues remain, there could be a problem with your DNS domain name-to-Kerberos realm mapping. This is especially apparent when the DNS domain name and Kerberos realm names do not match. Every time a client acquires a ticket to communicate with a Kerberized application server, it needs to determine the realm that the application server belongs to. This mapping is controlled by the *domain_realm* stanza in the */etc/krb5.conf* file, or TXT entries in the DNS (as described in the “DNS and Kerberos” section in Chapter 4). Failing to find an appropriate entry either in the configuration file or the DNS, the Kerberos libraries will try the host’s DNS domain name, converted to uppercase, as the realm name. Problems related to the domain name-to-Kerberos realm mapping can be diagnosed by narrowing down the problem to a particular problematic domain or host.

Unfortunately, this problem can cause mysterious failures with no error messages. The most direct way to find whether there are mismatched realm-DNS mappings is to check the KDC logs for the client’s default realm. If the client attempts to contact a Kerberized service that it thinks is in a different realm, it will attempt a cross-realm authentication to that other realm.

For example, we are on *slave.wedgie.org*, authenticated as *jgarman@WEDGIE.ORG* and attempting a Kerberized telnet to host *frebsd.wedgie.org*, which the slave thinks is in the *BOGUS.COM* realm, due to a configuration error. Therefore, it is trying to acquire service tickets for *host/slave.wedgie.org@BOGUS.COM*, but first needs to

acquire cross-realm tickets for BOGUS.COM. In this example, the following log entries are located in the log file of the KDC responsible for WEDGIE.ORG:

```
Feb 26 21:32:20 freebsd krb5kdc[520]: TGS_REQ (3 etypes {16 3 1}) 192.168.0.5(88
): UNKNOWN_SERVER: authtime 1046294981, jgarman@WEDGIE.ORG for krbtgt/BOGUS.COM
@WEDGIE.ORG, Server not found in Kerberos database
```

The solution to a DNS-to-realm mismatch is to ensure that all clients have the correct mappings in place. In this rather contrived example, the default behavior of Kerberos is sufficient, since all machines in the DNS domain *wedgie.org* belong to the WEDGIE.ORG Kerberos realm. However, other organizations may have a realm name different from their DNS domain name, and it is this situation that requires special care.

Server Hostname Misconfiguration

Undoubtedly, you'll find yourself in the following situation. You've just installed Kerberos 5; your KDC works, as you can acquire tickets for your user principal. You've dutifully established a host principal for your test application server, and you're ready to test the first application.

Your shell output looks something like this:

```
> hostname
freebsd.wedgie.org
> klist
Ticket cache: FILE:/tmp/krb5cc_p82191
Default principal: jgarman@WEDGIE.ORG

Valid starting   Expires         Service principal
01/29/03 04:52:21    01/29/03 10:30:11    krbtgt/WEDGIE.ORG@WEDGIE.ORG
```

```
Kerberos 4 ticket cache: /tmp/tkt1000
klist: You have no tickets cached
> ftp freebsd
Connected to localhost.
220 freebsd.wedgie.org FTP server (Version 5.60) ready.
334 Using authentication type GSSAPI; ADAT must follow
GSSAPI accepted as authentication type
GSSAPI error major: Miscellaneous failure
GSSAPI error minor: Server not found in Kerberos database
GSSAPI error: initializing context
GSSAPI authentication failed
334 Using authentication type KERBEROS_V4; ADAT must follow
KERBEROS_V4 accepted as authentication type
Kerberos V4 krb_mk_req failed: You have no tickets cached
Name (localhost:jgarman):
331 Password required for jgarman.
Password:
```

What just happened? It *should* work, after all. Let's double-check the principals in our Kerberos database:

```
% kadmin
Authenticating as principal jgarman/admin@WEDGIE.ORG with password.
Enter password:
kadmin: listprincs
K/M@WEDGIE.ORG
host/freebsd.wedgie.org@WEDGIE.ORG
ftp/freebsd.wedgie.org@WEDGIE.ORG
host/desktop.wedgie.org@WEDGIE.ORG
host/slave.wedgie.org@WEDGIE.ORG
imap/freebsd.wedgie.org@WEDGIE.ORG
ldap/freebsd.wedgie.org@WEDGIE.ORG
krbtgt/WEDGIE.ORG@WEDGIE.ORG
kadmin/admin@WEDGIE.ORG
kadmin/changepw@WEDGIE.ORG
kadmin/history@WEDGIE.ORG
jgarman@WEDGIE.ORG
jgarman/admin@WEDGIE.ORG
```

According to our Kerberos KDC, we have the correct service principal installed (specifically, *ftp/freebsd.wedgie.org@WEDGIE.ORG*). But still, ftp reports “Server not found in Kerberos database.”

A hint to the problem can be seen in the ftp output. Note that the first line of the ftp client's output reads:

```
Connected to localhost.
```

This means that the hostname “freebsd” resolves to the loopback address. Sure enough, our */etc/hosts* file contains the following line:

```
127.0.0.1 localhost localhost.wedgie.org freebsd freebsd.wedgie.org
```

Therefore, the ftp server receives the connection on the loopback interface. When the ftp daemon generates the service principal that it will use to validate the user's ticket, it performs an IP-to-name lookup on the interface on which it received the request. Therefore, if we examine the KDC request logs, we'll see a failed request for the nonexistent principal *ftp/localhost@WEDGIE.ORG*:

```
Jan 29 04:52:21 freebsd krb5kdc[35553]: TGS_REQ (3 etypes {16 3 1}) 192.168.0.5(88):
UNKNOWN_SERVER: authtime 1043800211, jgarman@WEDGIE.ORG for ftp/localhost@WEDGIE.
ORG, Server not found in Kerberos database
```

Once the */etc/hosts* file has been corrected with the correct IP-to-hostname mapping for our host, we get the intended result:

```
> ftp freebsd
Connected to freebsd.wedgie.org.
220 freebsd.wedgie.org FTP server (Version 5.60) ready.
334 Using authentication type GSSAPI; ADAT must follow
GSSAPI accepted as authentication type
GSSAPI authentication succeeded
Name (freebsd:jgarman):
232 GSSAPI user jgarman@WEDGIE.ORG is authorized as jgarman
```

This example illustrates the importance of consistent resolver and DNS information to the proper functioning of Kerberos. This problem can manifest itself in many ways, but all with the same root cause of incorrect data in DNS or local host databases. The particular scenario depicted above can occur with most Linux distributions, which, out of the box, alias the hostname of the machine to the machine's loopback address. Similar scenarios occur with Solaris machines, which, by default, create */etc/hosts* files that list the “short” hostname first, before the FQDN that Kerberos needs to form the correct service principal.

To ensure that Kerberized services can construct proper service principals, your local hosts database needs an entry like the following:

```
192.168.0.5  freebsd.wedgie.org  freebsd
```

Note that the hostname is mapped to the IP address of the appropriate interface (as opposed to the loopback IP address), and that the FQDN is given precedence over the short hostname.

Here's a similar error message that results from DNS and hostname misconfiguration:

```
Hostname cannot be canonicalized while verifying initial ticket
```

Essentially, this is caused when the application attempts to find its service principal name by performing a reverse lookup on the interface IP address that this request was received on. If that IP address does not resolve to a hostname, this message or the earlier “cannot find service principal” message will be generated.

Multi-homed hosts that have a different hostname for every interface also pose a problem for Kerberos. Depending on what interface a client's request arrives on, a different server hostname may be associated with that request. Therefore, each interface that has a different hostname has a different set of service principals based on that hostname, which can cause a problem if there is only one set of service principals defined for the host. It is recommended that multi-homed hosts have a single FQDN associated with all interfaces on the machine.

Encryption Type Mismatches

The extensible encryption type support in Kerberos 5 can result in some strange interactions when mixing different Kerberos 5 implementations. Most of the time the KDC can automatically determine the optimal set of encryption types for a given protocol exchange; however, sometimes it needs a little manual help.

This is necessary when you have keys stored for “stronger” encryption types in the KDC for a given service, but the service can only handle weaker encryption types. Since there's no direct communication between the service and the KDC, there is no way for the service to communicate its encryption-type support to the KDC. Instead, when the KDC returns a ticket to the client for use with the service, the KDC will use the strongest encryption type it has in its database for the service. Let's take an example.

We have a KDC *frebsd.wedgie.org*, running the MIT KDC, and an application server, *slave.wedgie.org*, running a Kerberized telnet daemon with the MIT client libraries. We've created a host keytab for *slave.wedgie.org*, and copied it into that host's */etc/krb5.keytab*. The KDC database and the keytab include keys for both triple DES and single DES encryption types, as illustrated below:

```
frebsd# /usr/local/sbin/kadmin
Authenticating as principal jgarman/admin@WEDGIE.ORG with password.
Enter password:
kadmin: getprinc host/slave.wedgie.org
Principal: host/slave.wedgie.org@WEDGIE.ORG
Expiration date: [never]
Last password change: Sun Nov 24 00:42:14 GMT 2002
Password expiration date: [none]
Maximum ticket life: 0 days 10:00:00
Maximum renewable life: 0 days 00:00:00
Last modified: Sun Nov 24 00:42:14 GMT 2002 (jgarman/admin@WEDGIE.ORG)
Last successful authentication: [never]
Last failed authentication: [never]
Failed password attempts: 0
Number of keys: 2
Key: vno 3, Triple DES cbc mode with HMAC/sha1, no salt
Key: vno 3, DES cbc mode with CRC-32, no salt
Attributes:
Policy: [none]
```

The keytab on *slave.wedgie.org* contains both encryption types, and consequently Kerberized telnet works fine between the two hosts:

```
> telnet -a -f slave
Trying 192.168.0.6...
Connected to slave.wedgie.org (192.168.0.6).
Escape character is '^]'.
[ Kerberos V5 accepts you as ``jgarman@WEDGIE.ORG'' ]
[ Kerberos V5 accepted forwarded credentials ]
Last login: Sun Feb 16 15:22:30 from 192.168.0.7
...
```

Now, let's see what happens when we erase the triple DES encryption key from slave's keytab. We'll use this to simulate a service that does not support triple DES encryption types. After we edit the keytab to remove the triple DES encryption key, we attempt another telnet into slave:

```
> telnet -a -f slave
Trying 192.168.0.6...
Connected to slave.wedgie.org (192.168.0.6).
Escape character is '^]'.
[ Kerberos V5 refuses authentication because telnetd: krb5_rd_req failed: Key table
entry not found ]
[ Kerberos V5 refuses authentication because telnetd: krb5_rd_req failed: Key table
entry not found ]
[ Trying KERBEROS4 ... ]
mk_req failed: You have no tickets cached
```

```
[ Trying KERBEROS4 ... ]  
mk_req failed: You have no tickets cached  
Password for jgarman:
```

This error message can be somewhat misleading; after all, we do have a key table entry for *host/slave.wedgie.org*; however, the keytab entry contains a single DES key, when the KDC has issued the client a service ticket encrypted with triple DES. Since the service can't find an appropriate encryption key to decrypt the ticket, it gives up and returns this error message.

The solution here is to identify which encryption types your service supports and ensure that the Kerberos database only contains those key encryption types for that service. For example, in this case, if our telnet server only supports single DES encryption, we need to remove the triple DES encryption key from the Kerberos KDC, so that the KDC will only issue tickets for that service encrypted with single DES. In general, all Kerberos 5 implementations must support single DES encryption, so if there is suspicion of encryption type incompatibilities, it is recommended that you recreate the relevant principals with a single DES encryption type. Unfortunately, this reduces the security of those principals.

To solve this problem, we delete the host key for *slave.wedgie.org*, re-create it with only a single DES encryption type, and extract that keytab onto the host. Since the KDC now only has a single DES encryption key available, it will encrypt the service ticket with the single DES key.

The process of creating principals with a subset of encryption types varies from implementation to implementation.