NAME

krb5_keytab_intro - The keytab handing functions

Kerberos Keytabs

See the library functions here: **Heimdal Kerberos 5 keytab handling functions**

Keytabs are long term key storage for servers, their equvalment of password files.

Normally the only function that useful for server are to specify what keytab to use to other core functions like krb5_rd_req() **krb5_kt_resolve()**, and **krb5_kt_close()**.

Keytab names

A keytab name is on the form type:residual. The residual part is specific to each keytab-type.

When a keytab-name is resolved, the type is matched with an internal list of keytab types. If there is no matching keytab type, the default keytab is used. The current default type is FILE.

The default value can be changed in the configuration file /etc/krb5.conf by setting the variable [defaults]default_keytab_name.

The keytab types that are implemented in Heimdal are:

- file store the keytab in a file, the type's name is FILE . The residual part is a filename. For compatibility with other Kerberos implemtation WRFILE and JAVA14 is also accepted. WRFILE has the same format as FILE. JAVA14 have a format that is compatible with older versions of MIT kerberos and SUN's Java based installation. They store a truncted kvno, so when the knvo excess 255, they are truncted in this format.

- keytab store the keytab in a AFS keyfile (usually /usr/afs/etc/KeyFile), the type's name is AFSKEYFILE. The residual part is a filename.

- memory The keytab is stored in a memory segment. This allows sensitive and/or temporary data not to be stored on disk. The type's name is MEMORY. Each MEMORY keytab is referenced counted by and opened by the residual name, so two handles can point to the same memory area. When the last user closes using **krb5_kt_close()** the keytab, the keys in they keytab is memset() to zero and freed and can no longer be looked up by name.

Keytab example

This is a minimalistic version of ktutil.

```
int
main (int argc, char **argv)
{
    krb5_context context;
    krb5_keytab keytab;
    krb5_kt_cursor cursor;
    krb5_keytab_entry entry;
    krb5_error_code ret;
    char *principal;

    if (krb5_init_context (&context) != 0)
        errx(1, ’krb5_context’);

    ret = krb5_kt_default (context, &keytab);
    if (ret)
        krb5_err(context, 1, ret, ’krb5_kt_default’);

    ret = krb5_kt_start_seq_get(context, keytab, &cursor);
    if (ret)
        krb5_err(context, 1, ret, ’krb5_kt_start_seq_get’);
    while((ret = krb5_kt_next_entry(context, keytab, &entry, &cursor)) == 0){
        krb5_unparse_name(context, entry.principal, &principal);
        printf(’principal: %s0, principal);
        free(principal);
        krb5_kt_free_entry(context, &entry);
    }
    ret = krb5_kt_end_seq_get(context, keytab, &cursor);
    if (ret)
        krb5_err(context, 1, ret, ’krb5_kt_end_seq_get’);
    ret = krb5_kt_close(context, keytab);
    if (ret)
        krb5_err(context, 1, ret, ’krb5_kt_close’);
    krb5_free_context(context);
    return 0;
}
```