KWallet - The KDE Wallet System

George Staikos <george@staikos.net>

08/24/03

"Better late than never... Better never than broken."

1 Why A Wallet?

Computer users have a very large amount of data to manage, some of which is sensitive. In particular, users typically have many passwords to manage. Remembering them is difficult, writing them down on paper or in a text file is insecure, and using tools such as PGP is tedious and inconvenient. The clean solution is to store the data in an encrypted package accessible by all applications and protected with a master password. This is typically known as a *keychain* or *wallet*, and facilitates the procedure of *single sign-on*. Apple implements one, Mozilla implements one, and many other organizations provide similar solutions.

2 High-Level Design

KDE provides one of the most tightly integrated desktop operating environments in existence. The implementation of a wallet should complement this integration. A single wallet for the entire system is ideal, allowing all applications to share the same data resources, facilitating easy editting, and making it easy to migrate this data to new machines. However, multiple access to the same file always results in synchronization issues. This is effectively unavoidable in KDE due to the nature of KIO slaves and the highly asynchronous I/O system. Thus a central wallet service was created to handle wallet file I/O, access control, caching, and even provide an easy way for non-KDE applications to access the wallet. Note that the architecture provides for multiple wallet files, though in practice most users will use one or two wallets at most. Ideal security would provide for one wallet for local passwords, and one wallet for network passwords and form data. The user may configure this ideal mode, but for simplicity the default is to store everything in one wallet.

Naturally the user should always be aware when a wallet is unlocked and informed as to which applications are accessing that wallet. The user should also be able to disconnect applications from a wallet if desired. Therefore when a wallet is opened, the KWallet service launches the KWallet Manager, a tray application which allows the user to monitor wallet activity and manage his wallets.

The contents of wallets are quite simple in nature, and very extensible for future enhancements. Each wallet consists of an unlimited number of unnested folders. Each folder has a name, and contents. The contents are a set of name-value pairs. The name may be any UTF-8 string, and the value may be any QByteArray. Since a QByteArray may contain literally any type of data, the application must be aware of the contents of a given entry in order to read it. For convenience, two value types have been predefined: Passwords (UTF-8 strings) and Maps (QMap<QString,QString>). These special types may be read or written through the corresponding specialized functions in the client API, and are identified as such in the wallet for type safety purposes.

3 File Format

The wallet files are portable, encrypted containers of data written out by Qt's QDataStream class. They consist of a header and a payload. The header is used to identify the file as a wallet file, as well as determine the file format revision. The payload is the encrypted portion and contains the data stream size, the data stream, and a data integrity hash. A whitening block is also contained in the payload to make it more difficult to break the encryption. The algorithms in use in the currentt version (0.0) of the file format are:

- 16 rounds blowfish in CBC mode with variable size key and one block initial random vector.

- SHA-1 hashing of the passphrase as well as hashing of the entire payload for data integrity purposes.

The file format is endian safe and compatible across architectures with different word sizes. This means that you can safely copy a file from one platform to another and access it without problems. The header contains an additional mechanism to determine if a file has been corrupted by an invalid transformation, such as an ftp transfer in ASCII mode between a DOS system and a Unix system. The two sequences "\n\r" and "\r\n" can be mangled by such an invalid transfer. The header also contains a mechanism to support multiple encryption and hashing algorithms for a given file format.

File Format Diagram:

Header	**16 bytes**
Magic String ("KWALLET\n\r\0\r\n")	
Format Version - Major (0)	
Format Version - Minor (0)	
Cipher Version (0 - CBC Blowfish)	
Hash Version (0 - SHA-1)	
Payload	**?? bytes**
Whitening block	8 bytes
Length of the data stream	32 bit BE
QDataStream output	?? bytes
Padding (random data)	?? bytes
Data hash	20 bytes

Note: the file format is subject to change until KDE 3.2.0 is released. After that time the format will remain compatible when the revision number is the same. After KDE 3.2.0 the revision number may change if changes to the format are required. Backwards compatibility with older wallets will always be provided.

4 Backend API

The backend API is considered private, with no installed headers. It provides the encryption, hashing, password manipulation, and low-level file I/O. Since it is subject to regular change, only the important algorithms will be outlined. When a wallet is opened, the backend API loads the entire file into memory as a map of maps of folders to keys to values. All temporary storage is overwritten to avoid leaving sensitive data in memory when the wallet is closed.

The wallet opening procedure consists of the following:

1. Hash the password.
2. Read in the file.
3. Check the magic header.
4. Check the version number and algorithm numbers.

5. Read in the file, decrypt, and hash it.
6. Compare the hash to determine data integrity.
7. Decode the datastream into a map.
8. Read in a string and a size_t representing the folder name and the number of entries in the folder.
9. Read in each entry as a string (key), long converted to an enumeration KWallet::Wallet::EntryType which represents the type of entry (Map, Password, etc), and a QByteArray representing the value.

The procedure for closing a wallet is similarly straightforward. The data is marshalled through QDataStream into a QByteArray which is then hashed and encrypted. The resulting size is computed and the file is written out with a header.

Entries in the backend are stored as pointers to KWallet::Entry objects. This allows pointer based manipulation (a bit more efficient) and makes it easier to track data for memory wiping purposes. No implicit copies are made.

5 Client API

KWallet was designed and redesigned four times in an effort to obtain the a solid foundation for the future. Much of this effort was spent on providing a clean client API. The client API consists of exactly one class, KWallet::Wallet, from the installed header kwallet.h and the installed library libkwalletclient. This class provides several static methods along with a rich list of member functions. To open a wallet, an application must request access, and if it is permitted (and the password is valid), a pointer to a KWallet::Wallet will be returned:

```
KWallet::Wallet *wallet =
KWallet::Wallet::openWallet("mywallet");
```

The wallet name in this example is "mywallet". In most cases, an application will not know the name of the wallet, but will instead use the helper functions KWallet::Wallet::LocalWallet() and KWallet::Wallet::NetworkWallet(). Typical usage of KWallet is similar to this:

```
if (!wallet->hasFolder("my folder")) {
    rc = wallet->createFolder("my folder");
    if (rc != 0) {
        // error
    }
}
rc = wallet->setFolder("my folder");
if (rc != 0) {
    // error
}
rc = wallet->readPassword("password key",

                            strbuf);

if (rc != 0) {
    // error
}
rc = wallet->writeMap("map key",

                      mapvariable);

if (rc != 0) {
    // error
}
```

Applications should generally store a pointer to the wallet which is initialized on first use and saved until exit. This will allow the application to regularly use a wallet without constantly prompting the user for a password. In order to detect if a wallet handle is invalidated, an application should connect to the walletClosed() signal. If an application needs to track changes to the wallet structure, the signals folderUpdated(), folderListUpdated(), and folderRemoved() are provided. The wallet may be closed (also called "locked" since the wallet is encrypted) with the lockWallet() method. It will be implicitly locked if the wallet object is deleted.

The keys used to store variables are arbitrary, but there will be a standard for certain keys to facilitate interoperability among applications. Standards should be defined for storing cookies, passwords, form data, and credit card numbers.

6 Service

The KWallet service provides synchronization and wallet file I/O. It is considered to be a *private* interface in that no application should talk to it directly, instead preferring the KWallet client API. However, the DCOP interface provided by the KWallet service is also the primary method available to non-KDE applications to use the KWallet system. The interface is implemented as a module in kded, the KDE Daemon. It can be accessed as the kwalletd object of the kded application. The present interface looks like this:

```
int open(QString wallet)
int close(QString wallet,bool force)
int close(int handle,bool force)
int deleteWallet(QString wallet)
bool isEnabled()
bool isOpen(QString wallet)
bool isOpen(int handle)
QStringList users(QString wallet)
void changePassword(QString wallet)
QStringList wallets()
QStringList folderList(int handle)
bool hasFolder(int handle,QString folder)
bool createFolder(int handle,
                          QString folder)
bool removeFolder(int handle,
                          QString folder)
QStringList entryList(int handle,
                          QString folder)
QByteArray readEntry(int handle,
        QString folder, QString key)
QByteArray readMap(int handle,
        QString folder, QString key)
QString readPassword(int handle,
        QString folder, QString key)
int renameEntry(int handle,
        QString folder, QString oldName,
                        QString newName)
int writeEntry(int handle,
        QString folder, QString key,
    QByteArray value, int entryType)
int writeEntry(int handle,
        QString folder, QString key,
                   QByteArray value)
int writeMap(int handle,
        QString folder, QString key,
                   QByteArray value)
int writePassword(int handle,
        QString folder, QString key,
                      QString value)
bool hasEntry(int handle,
        QString folder, QString key)
int entryType(int handle,
        QString folder, QString key)
int removeEntry(int handle,
        QString folder, QString key)
bool disconnectApplication(
                     QString wallet,
                QCString application)
```

In addition, the KWallet service controls access to wallets and requests passwords from the user when required. The password is cached inside the daemon as long as the wallet is open, and is overwritten and wiped from memory when the wallet is closed.

Wallet access is controlled via *handles* and DCOP application IDs. When an application successfully opens a wallet, a handle is issued and the application is tracked by the KWallet service. A handle is represented as a long integer, and is generated pseudorandomly. It is not meant to add hard security, only to protect against misbehaving applications. If the application unregisters from DCOP, the handle is automatically invalidated. By default, wallets are automatically closed when they are no longer in use.

7 Management Tool - kwalletmanager

The KWallet management tool serves two functions. First, it allows the user to see if any wallets are open, which wallets those are, and which applications are using each wallet. The user can force an application's handle for a given wallet closed from the user interface. Second, it allows the user to manage the wallets installed on the system, creating and deleting wallets and manipulating the contents of wallets.

When a wallet is opened, the kwalletmanager application is launched. A tray icon in the system tray indicates that a wallet is opened. When all wallets are again closed, the icon will change to a closed wallet to indicate the new situation. If the user clicks on the wallet, a window is displayed with an icon view showing all the wallets and their open/closed state. The context menu of each icon allows the user to:

- Create a new wallet.
- Open the wallet.
- Change the password of the wallet.
- Disconnect an application from the wallet.
- Force the wallet closed.
- Delete the wallet.

Wallets can also be dragged from the manager window. The drag will actually contain a URL pointing to the physical wallet file on disk. This allows the user to drag the wallet to a file browser window, copying, moving, or linking it as desired.

If a wallet is opened inside the wallet manager, a new window is displayed with the contents of the wallet. The window contains four panes, one showing a summary of the contents of the currentt folder, the next showing the folders contained in the wallet, the third showing the entries in the selected folder, and the fourth displaying the contents of a selected folder entry. Folders may be added or deleted, and selecting a folder will update the folder entry list and the summary display panes. Selecting a folder entry will update the entry contents pane and allow the user to edit that entry. Entries may also be created or deleted via the context menu for the folder contents.

All folders and entries may be dragged and dropped into other wallets or folders respectively. This allows a user to easily package up a new wallet for transfer to another environment. For instance, a new wallet could be created and copied onto a removable flash memory device. Important passwords could be transferred there, leaving unnecessary ones behind.

8 Configuration

KWallet contains a small configuration panel with several options for users to tune it to their individual preferences. Applications do not need to be aware of these settings, as they are transparently dealt with in the service and the client API. The default settings for KWallet are sufficient for most users.

9 Comparison to Other Systems

9.1 Apple Keychain

http://developer.apple.com/documentation/Security /Reference/keychainservices/keychainServRef/APIIndex.html

- 54 functions, 1 callback, 13 data types, lots of constants.
- Tight integration, global, easy for the user to use, nice GUI.
- More complicated than KWallet to develop with.
- Can browse the keychain without decrypting it, seeing entries.
- Provides very fine grained support for access control.
- Supports multiple keychains.

9.2 Q*Wallet

- Very restricted, no API or tight application integration.

- Seems to work with form data and site logins only.
- Only GUI provided.

9.3 Mozilla Wallet

- Specific to one application only.
- Geared heavily toward "form fill".
- GUI provided.

9.4 Microsoft Wallet

- Website has been pulled, uses .NET Passport after IE 5.01.
- Requires IE and Windows.
- Confusing API.
- Lots of proprietary language bindings!

9.5 KWallet

- Shared across all KDE applications, with ability for external applications to connect.
- Very simple to understand and use.
- Supports multiple wallets with folders.
- 30 functions, 4 signals, 2 data types, 4 constants.
- GUI provided.

10 Design Issues

- Why isn't protected memory used?
  - A virus that can read the memory of kded can also read the memory of konqueror when konqueror is connected to the wallet server and consequently use that to access the suid server anyways.
- Is using a socket instead of DCOP more secure?
  - Perhaps, but we lose flexibility, and is it really worth it?
- How do I access a wallet that is not installed?
  - The API does not provide for this at the moment. This may be fixed in the near future by providing KWallet::Client support for opening wallet files directly.

11 Who Should Use it?

- KIO (kpasswdserver)
- KHTML
- KMail
- KNode
- Kopete
- KCookieJar
- Any other application that stores passwords or sensitive user data.

At this time no application is activeely using KWallet, though preliminary patches are being developed for several of the items listed above.

12 Conclusion

While it was a long time coming, the wishlist entry #25994 and its 394 votes indicate that KWallet is definitely a desired feature for KDE. The implementation is simple, powerful, extensible, and easy to work with. As applications begin to utilize this service, it is expected that KDE will become a more secure desktop platform.