KSecret Service BoF

Valentin Rusu

Context

- KSecret Service development started several years ago
  - fd.o draft specified in collaboration with gnome
    - http://standards.freedesktop.org/secret-service/
  - gnome-keyring already implements this specification
  - KSecret Service have been put on hold during kdelibs splitting into KDE Framewworks
- meanwhile I took over maintainership of the KWallet so I now have a good grasp of the related issues along with user feedback

KWallet issues

Top slide

Security issues

- Client/Server architecture
  - Application → KWallet Framework → DBus → kwalletd
  - The DBus communication is not encrypted so sniffers could be used to reveal passwords
- No ACL
  - Every application can actually enumerate wallet contents

Usability issues

- When starting their KDE session, Network Manager, Owncloud client and other network-related agents need to get their password, so users see a rather bizarre prompt about KDE system wanting to get access to the KDE Wallet
- User's do not always get why the KWallet is there and what really it is
- KWallet Requires a separate main password
  - User's are constantly trying to deactivate KWallet because entering yet another password really annoys them

KWallet Manager

- KWallet Manager is somewhat designed for technical guys
- We have some bug reports about people worried that their colleagues could stole their passwords if they left away forgetting their computer unattended and with both their KDE session and the kwalletd unlocked

Kwallet Manager (continued)

- Questions:
  - is this tool really needed by common users?
  - Isn't it an advanced tool?
    - Only needed when someone forgot some application-stored password?
  - How to prevent it from being used by malicious users?
  - Should it have import/export functions?
    - For what purpose?
    - From/To what other password stores?

What's really needed?

Top slide

Getting out of the way

- Secrets should be transparently handled by the applications
- Many users do not really care how and where the secrets are stored. They just need the computer to remember them and correctly re-enter them when needed
- What should be done to keep this even more out of the way?

Being secure

- Some users are concerned about the security of these passwords, so they must be assured that getting out the passwords from a stolen computer won't be easy
  - Sure enough, system-wide encryption is a true option, but common users do not consider it

- Advanced users should be able:
  - to specify where the secrets are stored
  - sync their passwords on multiple devices ⇒ or should this be considered for all the users ?

What else?

- What else would you expect from KSecrets Service?

KSecret Service design

Top slide

Reduce DBus exchanges at a minimum

- Requiring a DBus session is a problem for setups where there's no dbus available
  - Android
  - No graphical session
- As stated above, DBus exchanges should be encrypted to prevent sneaking-in

Secrets unlocking

- pam_ksecrets pam module would derive two keys from user's password
  - The derived keys are then stored into the user session's kernel keyring
- ksecrets framework will offer the API KDE applications should use, instead of the KWallet API - more on this later
  - the framework will contain the decrypting library
    - It'll work even if a dbus session is not present
    - No security issues related to a daemon keeping secrets file opened around

Secrets file

- Encrypted with gcrypt using the keys prepared by the pam_ksecrets module
- The file structure will be something like this:

MAGIC | LENGTH | IV | ENCRYPTED_DATA | MAC_OF_ENCRYPTED

    - The Data is encrypted with the first key
    - The MAC is calculated using the second key

- Concurrency will be allowed by the use of a .lock file
    - A second lock file would be considered only if performances are to be optimised
    - It could stay in an owncloud synced folder

Changing the password

- The user's session password is used via pam_ksecrets
- When the user changes the password, pam_ksecrets gets notified
    - If the secrets file is not locked, the module would allow changing the password
    - A new key-pair will be derived and the file will be re-encrypted with it
    - The old secrets file will be kept as a backup. It will decrypt with the old password.

KSecrets Manager

- This is the KWallet Manager equivalent
- It's still needed in order to let users retrieve their passwords
- It also uses the library given by the API, so it would not expose the KWallet Manager security issues
- The user interface will be designed with the help of the VDG in order to make it more user-friendly
- KSecrets Manager is an advanced tool and should be a settings module. It would be analogous to the cookies management KCM.

A word about the ACL

- The secrets file will be split into "collections"
    - Each collection will be dedicated to the original application that created it
    - Only the original application will be allowed to access the collection
    - No dialog will pop-up asking for collection unlock. However, a dialog should pop-up only when ACL warnings/errors are encountered

- Should this be managed via the KSecrets Manager?
  - Either ways, this application needs to receive "special permissions"
    - How to ensure that only KSecrets Manager is the one having the rights to handle and read the secrets data?

Proposed architecture

- KDE App → KSecrets Framework → Secrets file
- Non KDE App → DBus → ksecretsd → KSecrets Framework → Secrets file
  - This should be offered via system alternatives in order to allow users switch between ksecretsd and gnome-keyring
  - the fd.o draft should specify where to config the alternative
- KSecrets Manager → KSecrets Framework → Secrets file
  - Should it provide a QML interface? Or better stick with the QWidget infrastructure?
    - Either ways, the model will be separated from the UI

NOTE The API should be async from the start to provide for dbus integration.

The API

- Hosted into a new "KSecrets Framework"
- Should be compatible with the freedesktop.org interface
  - A future release will provide ability to connect to the freedesktop.org interface, e.g. to the gnome-keyring
- Will be implemented as a library and would not call the daemon at all
- Will expect finding the encryption keys into the kernel keyring
- Will also be used by the ksecrets

Migrating from KWallet

Top slide

Scenario

- First, get the pam_ksecrets module configured on user systems
  - This should be activated beforehand
- A migration agent will be triggered upon session start asking user permission to proceed
- KWallet API should be modified in order to first try to get the secrets from the KSecrets and only to get it from kwalletd if not found
  - The same should be done for the other way around, when writing, so new items should get into KSecrets file
  - The migration should be done entirely done per application and NOT secret by secret.
- Applications ported from KWallet to KSecrets will specify a flag that would provide for a dialog inviting user to select where was the old informations in KWallet
  - The KSecrets API would directly connect to kwalletd to get that information, without using KWallet, to avoid depending on it

NOTE The KSecrets API handles secrets grouped into collections. KWallet, on the other hand, gets information about the opening application by the means of the parameters given upon openWallet. So perhaps it'll be a good idea to try to use the process name for collection manipulation. And KSecrets API should automatically construct collection name from the calling process name. If doing that, the new API will automatically find secrets written by callin KWallet API.

Roadmap

- [code currently in plaground/util/ksecrets]

1. pam_ksecrets
   - This one is already done
2. KSecrets Framework
   - Will contain ksecrets-cli ⇒ tool useful for script addicts
3. KWallet API adapting for KSecrets
4. KSecrets Manager
5. ksecretsd

- This would allow GNOME applications integrate with KDE

6. KSecrets Framework interface with gnome-keyring